User's Manual, V1.0, Jan. 2002

TC1765

System Units

32-Bit Single-Chip Microcontroller

32bit

Microcontrollers

Infineon technologies

Never stop thinking.

Edition 2002-01

Published by Infineon Technologies AG,
St.-Martin-Strasse 53,
D-81541 München, Germany

© Infineon Technologies AG 2002.
All Rights Reserved.

Attention please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Infineon Technologies is an approved CECC manufacturer.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office in Germany or our Infineon Technologies Representatives worldwide.

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

TC1765

System Units

32-Bit Single-Chip Microcontroller

Microcontrollers

Never stop thinking.

TC1765 System Units User's Manual

Revision History:	**2002-01**	V1.0
Previous Versions:	none	

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:

mcdocu.comments@infineon.com

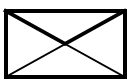


Table of Contents **Page**

1	**Introduction**	1-1
1.1	About this Document	1-1
1.1.1	Related Documentations	1-1
1.1.2	Textual Conventions	1-1
1.1.3	Reserved, Undefined, and Unimplemented Terminology	1-3
1.1.4	Register Access Modes	1-4
1.1.5	Abbreviations	1-4
1.2	System Architecture Features of the TC1765	1-6
1.3	Block Diagram	1-10
1.4	On-Chip Peripheral Units of the TC1765	1-11
1.4.1	Serial Interfaces	1-12
1.4.1.1	Asynchronous/Synchronous Serial Interfaces	1-12
1.4.1.2	High-Speed Synchronous Serial Interfaces	1-14
1.4.1.3	TwinCAN Interface	1-16
1.4.2	Timer Units	1-18
1.4.2.1	General Purpose Timer Unit	1-18
1.4.2.2	General Purpose Timer Array	1-20
1.4.3	Analog-to-Digital Converters	1-23
1.5	Pin Definitions and Functions	1-25
2	**TC1765 Processor Architecture**	2-1
2.1	Central Processing Unit	2-2
2.1.1	Instruction Fetch Unit	2-3
2.1.2	Execution Unit	2-4
2.1.3	General Purpose Register File	2-5
2.1.4	Program State Registers	2-6
2.1.5	Data Types	2-7
2.1.6	Addressing Modes	2-7
2.1.7	Instruction Formats	2-7
2.1.8	Tasks and Contexts	2-7
2.1.8.1	Upper and Lower Contexts	2-8
2.1.8.2	Context Save Areas	2-9
2.1.8.3	Fast Context Switching	2-9
2.1.9	Interrupt System	2-10
2.1.10	Trap System	2-10
2.1.11	Protection System	2-11
2.1.11.1	Permission Levels	2-11
2.1.11.2	Memory Protection Model	2-11
2.1.11.3	Watchdog Timer and ENDINIT Protection	2-12
2.1.12	Reset System	2-12
2.2	Processor Registers	2-13
2.2.1	Program State Information Registers	2-16
2.2.1.1	Program Counter (PC)	2-16

Table of Contents **Page**

2.2.1.2 Program Status Word (PSW) 2-17
2.2.1.3 Previous Context Information Register (PCXI) 2-21
2.2.2 Context Management Registers 2-23
2.2.2.1 Free Context List Head Pointer (FCX) 2-23
2.2.2.2 Previous Context Pointer (PCX) 2-24
2.2.3 Free Context List Limit Pointer (LCX) 2-25
2.2.4 Stack Management 2-26
2.2.4.1 Interrupt Stack Pointer (ISP) 2-26
2.2.5 Interrupt and Trap Control 2-27
2.2.5.1 Interrupt Vector Table Pointer (BIV) 2-27
2.2.5.2 Trap Vector Table Pointer (BTV) 2-28
2.2.6 System Control Register 2-29
2.2.7 Memory Protection Registers 2-30
2.2.8 Debug Registers 2-30
2.2.9 CSFR Address Table 2-31
2.3 Instruction Set Overview 2-34
2.3.1 Arithmetic Instructions 2-34
2.3.1.1 Integer Arithmetic 2-35
2.3.1.2 DSP Arithmetic 2-42
2.3.2 Compare Instructions 2-46
2.3.3 Bit Operations 2-50
2.3.4 Address Arithmetic 2-52
2.3.5 Address Comparison 2-53
2.3.6 Branch Instructions 2-54
2.3.6.1 Unconditional Branch 2-54
2.3.6.2 Conditional Branch 2-55
2.3.6.3 Loop Instructions 2-56
2.3.7 Load and Store Instructions 2-57
2.3.7.1 Load/Store Basic Data Types 2-57
2.3.7.2 Load Bit 2-59
2.3.7.3 Store Bit and Bit Field 2-59
2.3.8 Context Related Instructions 2-60
2.3.8.1 Context Saving and Restoring 2-60
2.3.8.2 Context Loading and Storing 2-60
2.3.9 System Instructions 2-61
2.3.9.1 System Call 2-61
2.3.9.2 Synchronization Primitives 2-61
2.3.9.3 Access to the Core Special Function Registers 2-62
2.3.9.4 Enabling/Disabling the Interrupt System 2-62
2.3.9.5 RET and RFE 2-63
2.3.9.6 Trap Instructions 2-63
2.3.9.7 No Operation 2-63

Table of Contents **Page**

2.3.10 16-Bit Instructions . . . 2-63
2.4 CPU Pipelines . . . 2-64
2.4.1 CPU Pipeline Overview . . . 2-64
2.4.2 Integer and Load/Store Pipelines . . . 2-64
2.4.3 Loop Pipeline . . . 2-66
2.4.4 Context Operations . . . 2-67

3 Clock System . . . 3-1
3.1 Clock Generation Unit . . . 3-3
3.1.1 Oscillator Circuit . . . 3-4
3.1.2 Phase-Locked Loop (PLL) . . . 3-5
3.1.2.1 N-Divider . . . 3-5
3.1.2.2 VCO Frequency Ranges . . . 3-5
3.1.2.3 Lock Detection . . . 3-5
3.1.2.4 K-Divider . . . 3-5
3.1.2.5 Clock Source Control . . . 3-6
3.1.3 Enable/Disable Control . . . 3-6
3.1.4 Determining the System Clock Frequency . . . 3-7
3.1.4.1 PLL Bypass Operation . . . 3-7
3.1.4.2 PLL Operation . . . 3-7
3.1.5 PLL Clock Control and Status Register . . . 3-9
3.1.6 Startup Operation . . . 3-10
3.1.7 PLL Loss of Lock Operation . . . 3-10
3.2 Power Management and Clock Gating . . . 3-12
3.2.1 Clock Control . . . 3-13
3.2.2 Module Clock Generation . . . 3-14
3.2.3 Clock Control Registers . . . 3-15
3.2.4 CLC Register Implementations . . . 3-20

4 System Control Unit . . . 4-1
4.1 Overview . . . 4-1
4.2 Registers Overview . . . 4-2
4.3 SCU Control Register . . . 4-3
4.4 Trace Control . . . 4-7
4.5 Trace Status Register . . . 4-8
4.6 Identification Registers . . . 4-9

5 Reset and Boot Operation . . . 5-1
5.1 Overview . . . 5-1
5.2 Reset Registers . . . 5-2
5.2.1 Reset Status Register (RST_SR) . . . 5-3
5.2.2 Reset Request Register (RST_REQ) . . . 5-5
5.3 Reset Operations . . . 5-7
5.3.1 Power-On Reset . . . 5-7

Table of Contents **Page**

5.3.2 External Hardware Reset 5-7
5.3.3 Software Reset 5-8
5.3.4 Watchdog Timer Reset 5-9
5.3.4.1 Watchdog Timer Reset Lock 5-9
5.3.4.2 Deep-Sleep Wake-Up Reset 5-10
5.3.5 State of the TC1765 after Reset 5-10
5.4 Booting Scheme 5-12
5.4.1 Hardware Booting Scheme 5-12
5.4.2 Software Booting Scheme 5-12
5.4.3 Boot Options 5-13
5.4.4 Boot Configuration Handling 5-14
5.4.5 Normal Boot Options 5-14
5.4.6 Debug Boot Options 5-14
5.4.7 Booting from Boot ROM 5-15

6 Power Management 6-1
6.1 Power Management Overview 6-1
6.2 Power Management Control Registers 6-3
6.2.1 Power Management Control Register PMG_CON 6-4
6.2.2 Power Management Control and Status Register PMG_CSR 6-5
6.3 Power Management Modes 6-6
6.3.1 Idle Mode 6-6
6.3.2 Sleep Mode 6-6
6.3.2.1 Entering Sleep Mode 6-6
6.3.2.2 TC1765 State During Sleep Mode 6-7
6.3.2.3 Exiting Sleep Mode 6-7
6.3.3 Deep Sleep Mode 6-7
6.3.3.1 Entering Deep Sleep Mode 6-8
6.3.3.2 TC1765 State During Deep Sleep Mode 6-8
6.3.3.3 Exiting Deep Sleep Mode 6-8
6.3.3.4 Exiting Deep Sleep Mode With A Power-On Reset Signal 6-9
6.3.3.5 Exiting Deep Sleep Mode With an $\overline{\text{NMI}}$ Signal 6-9
6.3.4 Summary of TC1765 Power Management States 6-10

7 Memory Map 7-1
7.1 TC1765 Address Map 7-2
7.2 Memory Segment 15 - Peripheral Units 7-5

8 Program Memory Unit 8-1
8.1 Memories Controlled by PMU 8-2
8.2 Scratch-Pad RAM, SPRAM 8-3
8.3 Instruction Cache, ICACHE 8-3
8.3.1 Cache Organization 8-4
8.3.2 Cache Bypass Control 8-4

Table of Contents **Page**

8.3.3 Refill Buffer 8-4
8.3.4 Refill Sequence for Cache and Refill Buffer 8-5
8.3.5 Cache Flush Operation 8-5
8.4 External Code Fetches via External Bus Interface Unit 8-5
8.5 Boot ROM 8-6
8.5.1 Bootstrap Loader Support 8-6
8.6 PMU Registers 8-7
8.6.1 PMU Control Register 8-8
8.6.2 External Instruction Fetch Control Register 8-10

9 Data Memory Unit 9-1
9.1 DMU Trap Generation 9-3
9.1.1 FPI Bus Error 9-3
9.1.2 Range Error 9-4
9.1.3 DMU Register Access Error 9-4
9.1.4 Cache Management Error 9-4
9.2 Overlay Functionality 9-5
9.2.1 Redirection From External Code to Internal Data Memory 9-6
9.2.2 Redirection From External Code to External Data Memory 9-7
9.2.3 Redirection from External Code via $\overline{\text{CODE}}$ to External Data Memory . 9-8
9.3 DMU Registers 9-10
9.3.1 Control Register 9-11
9.3.2 Synchronous Trap Flag Register 9-12
9.3.3 Asynchronous Trap Flag Register 9-13
9.3.4 Overlay Functionality Registers 9-14

10 Direct Memory Access Controller (DMA) 10-1
10.1 DMA Controller Kernel Description 10-2
10.1.1 Definition of Terms 10-3
10.1.2 DMA Principle 10-4
10.1.3 DMA Block Diagram 10-5
10.1.4 DMA Operation Functionality 10-6
10.1.5 DMA Channel Configuration 10-7
10.1.5.1 DMA Transfer Triggering 10-7
10.1.5.2 DMA Channel Operation Mode 10-7
10.1.5.3 DMA Transfer Parameters 10-12
10.1.5.4 DMA Bus Bandwidth Limitation 10-13
10.1.5.5 Error Conditions 10-13
10.1.5.6 Channel Reset Operation 10-13
10.1.6 Request Assignment Units 0 and 1 10-14
10.1.7 Request Assignment Unit 2 10-15
10.2 DMA Module Kernel Registers 10-17
10.3 DMA Module Implementation 10-28

Table of Contents **Page**

10.3.1 Interfaces of the DMA Module 10-28
10.3.1.1 Request Assignment Unit 0/1 Input Connections 10-29
10.3.1.2 Request Assignment Unit 2 Input Connections 10-30
10.3.2 DMA Module Related External Registers 10-31
10.3.2.1 Clock Control Register 10-32
10.3.2.2 Interrupt Registers 10-33
10.3.3 DMA Register Address Ranges 10-34

11 Memory Protection System 11-1
11.1 Memory Protection Overview 11-1
11.2 Memory Protection Registers 11-3
11.2.1 PSW Protection Fields 11-7
11.2.2 Data Memory Protection Register 11-11
11.2.3 Code Memory Protection Register 11-14
11.3 Sample Protection Register Set 11-17
11.4 Memory Access Checking 11-18
11.4.1 Permitted versus Valid Accesses 11-18
11.4.2 Crossing Protection Boundaries 11-19

12 Parallel Ports and Pins 12-1
12.1 General Port Operation 12-1
12.2 Port Registers 12-4
12.2.1 Data Output Register 12-6
12.2.2 Data Input Register 12-7
12.2.3 Direction Control Register 12-8
12.2.4 Port Open Drain Control Register 12-9
12.2.5 Input Configuration Register 12-10
12.2.6 Pull-Up/-Down Device Control 12-11
12.2.7 Output Characteristics Control Register 12-13
12.2.8 Alternate Port Functions 12-14
12.2.8.1 Alternate Input Functions 12-14
12.2.8.2 Alternate Output Functions 12-14
12.3 Port 0 12-15
12.3.1 Features 12-15
12.3.2 Registers 12-15
12.4 Port 1 12-23
12.4.1 Features 12-23
12.4.2 Registers 12-23
12.5 Port 2 12-25
12.5.1 Features 12-25
12.5.2 Registers 12-25
12.6 Port 3 12-27
12.6.1 Features 12-27

Table of Contents **Page**

12.6.2 Registers 12-27
12.7 Port 4 12-29
12.7.1 Features 12-29
12.7.2 Registers 12-29
12.8 Port 5 12-31
12.8.1 Features 12-31
12.8.2 Registers 12-31
12.9 Trace Port (TC1765T only) 12-34
12.10 External Bus Interface 12-34
12.11 ADC Analog Input Port 12-35
12.12 Electrical Characteristics 12-35

13 External Bus Unit 13-1
13.1 Overview 13-2
13.2 EBU Feature Summary 13-3
13.3 Basic EBU Operation 13-4
13.4 EBU Signal Description 13-5
13.4.1 EBU Clock Output, ECOUT 13-6
13.4.2 EBU Clock Input, ECIN 13-6
13.4.3 Address Bus, A[23:0] 13-6
13.4.4 Data Bus, D[31:0] 13-6
13.4.5 Read/Write Strobes, $\overline{RD}$ and RD/$\overline{WR}$ 13-6
13.4.6 Byte Control Signals, $\overline{BCx}$ 13-6
13.4.7 Variable Wait State Control, Burst Abort $\overline{WAIT}$/$\overline{IND}$ 13-7
13.4.8 Chip Select Lines, $\overline{CSx}$ 13-7
13.4.9 Instruction Fetch Indication Signal, $\overline{CODE}$ 13-8
13.4.10 Emulation/Instrumentation Support Signal $\overline{CSEMU}$/$\overline{CSOVL}$ 13-8
13.5 EBU Boot Process 13-9
13.6 FBU Operation for FPI Bus Based Transfers 13-12
13.7 FBU Memory Map 13-12
13.7.1 FBU External Address Regions ($\overline{CSx}$) 13-12
13.7.1.1 Address Region Selection 13-13
13.7.1.2 Address Region Parameters 13-16
13.7.2 Driver Turn-Around Wait States 13-17
13.7.3 Data Width of External Devices 13-19
13.7.4 Basic Access Timing 13-21
13.7.5 Variable Wait State Control, Burst Abort $\overline{WAIT}$/$\overline{IND}$ 13-25
13.7.6 Execution of Transfer Requests to the FBU 13-26
13.7.7 Atomic Read-Modify-Write Accesses 13-26
13.8 Emulation Support 13-27
13.8.1 Emulation Boot and Emulation Memory 13-27
13.8.2 Overlay Memory 13-27
13.8.3 Usage of $\overline{CSEMU}$/$\overline{CSOVL}$ 13-29

Table of Contents **Page**

13.9 BIFU Operation for Burst Mode Instruction Fetches 13-30
13.9.1 Basic Functions 13-30
13.9.2 External Instruction Fetch Control Register 13-32
13.9.3 Cycle Definitions of Burst Mode Timing 13-34
13.10 Arbitration between EBU and PMU/BIFU for External Accesses 13-42
13.11 EBU Registers 13-43
13.11.1 Clock Control Register 13-44
13.11.2 Global Control Register 13-45
13.11.3 Address Select Registers 13-46
13.11.4 Bus Configuration Registers 13-47
13.11.5 Emulator Configuration Register 13-50
13.11.6 Emulator Bus Configuration Register 13-51
13.11.7 Emulator Address Select Register 13-54
13.11.8 EBU Register Address Range 13-55

14 Interrupt System 14-1
14.1 Overview 14-1
14.2 External Interrupts 14-3
14.3 Service Request Nodes 14-3
14.3.1 Service Request Control Registers 14-3
14.3.1.1 Service Request Flag (SRR) 14-5
14.3.1.2 Request Set and Clear Bits (SETR, CLRR) 14-5
14.3.1.3 Enable Bit (SRE) 14-5
14.3.1.4 Service Request Flag (SRR) 14-6
14.3.1.5 Type-of-Service Control (TOS) 14-6
14.3.1.6 Service Request Priority Number (SRPN) 14-6
14.4 Interrupt Control Unit (ICU) 14-7
14.4.1 ICU Interrupt Control Register (ICR) 14-7
14.4.1.1 Operation of the Interrupt Control Unit (ICU) 14-9
14.5 Arbitration Process 14-11
14.5.1 Controlling the Number of Arbitration Cycles 14-11
14.5.2 Controlling the Duration of Arbitration Cycles 14-12
14.6 Entering an Interrupt Service Routine 14-12
14.7 Exiting an Interrupt Service Routine 14-13
14.8 Interrupt Vector Table 14-14
14.9 Usage of the TC1765 Interrupt System 14-17
14.9.1 Spanning Interrupt Service Routines Across Vector Entries 14-17
14.9.2 Configuring Ordinary Interrupt Service Routines 14-18
14.9.3 Interrupt Priority Groups 14-18
14.9.4 Splitting Interrupt Service Across Different Priority Levels 14-19
14.9.5 Using Different Priorities for the same Interrupt Source 14-20
14.9.6 Software Initiated Interrupts 14-21
14.9.7 Interrupt Priority 1 14-21

Table of Contents **Page**

14.10 CPU Service Request Nodes 14-22
14.11 Service Request Register Table 14-23

15 Trap System 15-1
15.1 Trap System Overview 15-1
15.2 Trap Classes 15-3
15.2.1 Synchronous Traps 15-5
15.2.2 Asynchronous Traps 15-5
15.2.3 Hardware Traps 15-5
15.2.4 Software Traps 15-5
15.2.5 Trap Descriptions 15-6
15.3 Trap Vector Table 15-10
15.3.1 Entering a Trap Service Routine 15-11
15.4 Non-Maskable Interrupt 15-12
15.4.1 NMI Status Register 15-12
15.4.2 External $\overline{\text{NMI}}$ Input 15-13
15.4.3 Phase-Locked Loop NMI 15-13
15.4.4 Watchdog Timer NMI 15-14

16 FPI Bus and Bus Control 16-1
16.1 FPI Bus Overview 16-1
16.2 Bus Control Unit 16-4
16.2.1 FPI Bus Arbitration 16-5
16.2.1.1 Arbitration Priority 16-5
16.2.1.2 Bus Starvation Protection 16-6
16.2.2 Error Handling 16-6
16.2.3 BCU Power Saving Mode 16-9
16.2.4 BCU Registers 16-10
16.2.4.1 BCU Control Register 16-11
16.2.4.2 BCU Debug Registers 16-12
16.2.4.3 BCU Service Request Control Register 16-15

17 System Timer 17-1
17.1 Overview 17-1
17.2 Kernel Functions 17-1
17.3 Kernel Registers 17-4
17.4 External Register 17-7
17.5 STM Register Address Ranges 17-8

18 Watchdog Timer 18-1
18.1 Watchdog Timer Overview 18-1
18.2 Features of the Watchdog Timer 18-2
18.3 The EndInit Function 18-3
18.4 Watchdog Timer Operation 18-5

Table of Contents **Page**

18.4.1 WDT Register Overview 18-6
18.4.2 Modes of the Watchdog Timer 18-7
18.4.2.1 Time-Out Mode 18-8
18.4.2.2 Normal Mode 18-8
18.4.2.3 Disable Mode 18-8
18.4.2.4 Prewarning Mode 18-9
18.4.3 Password Access to WDT_CON0 18-10
18.4.4 Modify Access to WDT_CON0 18-11
18.4.5 Term Definitions for WDT_CON0 Accesses 18-12
18.4.6 Detailed Descriptions of the WDT Modes 18-13
18.4.6.1 Time-Out Mode Details 18-13
18.4.6.2 Normal Mode Details 18-14
18.4.6.3 Disable Mode Details 18-15
18.4.6.4 Prewarning Mode Details 18-16
18.4.6.5 WDT Operation During Power-Saving Modes 18-17
18.4.6.6 WDT Operation in OCDS Suspend Mode 18-17
18.4.7 Determining WDT Periods 18-18
18.4.7.1 Time-out Period 18-19
18.4.7.2 Normal Period 18-20
18.4.7.3 WDT Period During Power-Saving Modes 18-21
18.5 Handling the Watchdog Timer 18-22
18.5.1 System Initialization 18-22
18.5.2 Re-opening Access to Critical System Registers 18-23
18.5.3 Servicing the Watchdog Timer 18-23
18.5.4 Handling the User-Definable Password Field 18-24
18.5.5 Determining the Required Values for a WDT Access 18-27
18.6 Watchdog Timer Registers 18-28
18.6.1 Watchdog Timer Control Register 0 18-29
18.6.2 Watchdog Timer Control Register 1 18-31
18.6.3 Watchdog Timer Status Register 18-32

19 On-Chip Debug Support 19-1
19.1 TriCore CPU Debug Support 19-2
19.1.1 Basic Concepts 19-2
19.1.2 Debug Event Generation 19-2
19.1.2.1 External Debug Break Input 19-3
19.1.2.2 Software Debug Event Generation 19-3
19.1.2.3 Execution of a MTCR or MFCR Instruction 19-3
19.1.2.4 Debug Event Generation from Debug Triggers 19-4
19.1.3 Debug Triggers 19-4
19.1.3.1 Protection Mechanism 19-4
19.1.3.2 Combination of Triggers 19-5
19.1.4 Actions Taken on a Debug Event 19-6

Table of Contents **Page**

19.1.4.1 Assert an External Pin $\overline{\text{BRKOUT}}$ 19-6
19.1.4.2 Halt 19-6
19.1.4.3 Breakpoint Trap 19-6
19.1.4.4 Software Breakpoint 19-7
19.1.5 OCDS Registers 19-8
19.2 Trace Module 19-17
19.2.1 Overview 19-17
19.2.2 Pipeline Status Signals 19-17
19.2.2.1 Synchronizing with the Status and Indirect Streams 19-19
19.2.3 Indirect Addresses 19-20
19.2.3.1 Indirect Sync 19-20
19.2.3.2 Example 19-22
19.2.4 Trace Output Control (TC1765T only) 19-23
19.3 DMA Controller Trace 19-24
19.4 Debugger Interface (Cerberus) 19-24
19.4.1 RW Mode 19-26
19.4.1.1 Entering RW Mode 19-26
19.4.1.2 Data Type Support 19-26
19.4.1.3 FPI Bus Master Interface 19-26
19.4.2 Communication Mode 19-27
19.4.3 System Security 19-27
19.4.4 Triggered Transfers 19-27
19.4.4.1 Tracing of Memory Locations 19-27
19.4.5 Power Saving 19-28
19.4.6 Registers 19-28
19.4.6.1 IOCONF Register 19-29
19.4.6.2 IOSR Register 19-31
19.4.6.3 JDP_ID Register 19-32
19.4.6.4 TRADDR Register 19-32
19.4.6.5 IOADDR, COMDATA and RWDATA Registers 19-32
19.5 Concurrent Debugging 19-33
19.6 OCDS Register Address Ranges 19-33

20 Register Overview 20-1
20.1 Segments 0 - 14 20-2
20.1.1 Address Map 20-2
20.1.2 Registers 20-4
20.2 Segment 15 (Peripheral Units) 20-8
20.2.1 Address Map 20-8
20.2.2 Registers 20-11

Table of Contents **Page**

21 Index 21-1
21.1 Keyword Index 21-1
21.2 Register Index 21-7

1 Introduction

This User's Manual describes the Infineon TC1765, a 32-bit microcontroller DSP, based on the Infineon TriCore Architecture. The device name TC1765 refers to both versions, TC1765N — the standard version, and TC1765T — the extended debugging version.

1.1 About this Document

This document is designed to be read primarily by design engineers and software engineers who need a detailed description of the interactions of the TC1765 functional units, registers, instructions, and exceptions.

This TC1765 User's Manual describes the features of the TC1765 with respect to the TriCore Architecture. Where the TC1765 directly implements TriCore architectural functions, this manual simply refers to those functions as features of the TC1765. In all cases where this manual describes a TC1765 feature without referring to the TriCore Architecture, this means that the TC1765 is a direct embodiment the TriCore Architecture.

Where the TC1765 implements a subset of TriCore architectural features, this manual describes the TC1765 implementation, and then describes how it differs from the TriCore Architecture. For example, where the TriCore Architecture specifies up-to four Memory Protection Register Sets, the TC1765 implements but two. Such differences between the TC1765 and the TriCore Architecture are documented in the text covering each such subject.

1.1.1 Related Documentations

A complete description of the TriCore architecture is found in the document titled "TriCore Architecture Manual". The architecture of the TC1765 is described separately this way because of the configurable nature of the TriCore specification: different versions of the architecture may contain a different mix of systems components. The TriCore architecture, however, remains constant across all derivative designs in order to preserve compatibility.

Additionally to this "TC1765 System Units User's Manual", a second document, the "TC1765 Peripheral Units User's Manual", is available. These two User's Manuals together with the "TriCore Architecture Manual" are required for the understanding the complete TC1765 microcontroller functionality.

Implementation-specific details such as electrical characteristics and timing parameters of the TC1765 can be found in the "TC1765 Data Sheet".

1.1.2 Textual Conventions

This document uses the following textual conventions for named components of the TC1765:

- Functional units of the TC1765 are given in plain UPPER CASE. For example: “The EBU provides an interface to external peripherals”.
- Pins using negative logic are indicated by an overbar. For example: “The external reset pin, $\overline{\text{HDRST}}$, has a double-function.”.
- Bit fields and bits in registers are in general referenced as “Register name.Bit field” or “Register name.Bit”. For example: “The Current CPU Priority Number bit field ICR.CCPN is cleared”. Most of the register names contain a module name prefix, separated by a underscore character “_” from the real register name (for example, “ASC0_CON”, where “ASC0” is the module name prefix, and “CON” is the real register name). In chapters describing peripheral modules the real register name is referenced also as kernel register name.
- Variables used to describe sets of processing units or registers appear in mixed-case font. For example, register name “MSGCFGn” refers to multiple “MSGCFG” registers with variable n. The bounds of the variables are always given where the register expression is first used (for example, “n = 31 - 0”), and is repeated as needed in the rest of the text.
- The default radix is decimal. Hexadecimal constants are suffixed with a subscript letter “H”, as in 100_H. Binary constants are suffixed with a subscript letter “B”, as in: 111_B.
- When the extent of register fields, groups of signals, or groups of pins are collectively named in the body of the document, they are given as “NAME[A:B]”, which defines a range for the named group from B to A. Individual bits, signals, or pins are given as “NAME[C]” where the range of the variable C is given in the text. For example: CFG[2:0], and TOS[0].
- Units are abbreviated as follows:
  - **MHz** = Megahertz
  - **µs** = Microseconds
  - **kBaud, kBit** = 1000 characters/bits per second
  - **MBaud, MBit** = 1,000,000 characters/bits per second
  - **KByte** = 1024 bytes of memory
  - **MByte** = 1048576 bytes of memory

  In general, the *k* prefix scales a unit by 1000 whereas the K prefix scales a unit by 1024. Hence, the KByte unit scales the expression preceding it by 1024. The kBaud unit scales the expression preceding it by 1000. The M prefix scales by 1,000,000 or 1048576, and µ scales by .000001. For example, 1 KByte is 1024 bytes, 1 MByte is 1024×1024 bytes, 1 kBaud/kBit are 1000 characters/bits per second, 1 MBaud/MBit are 1000000 characters/bits per second, and 1 MHz is 1,000,000 Hz.
- Data format quantities are defined as follows:
  - **Byte** = 8-bit quantity
  - **Half-word** = 16-bit quantity
  - **Word** = 32-bit quantity
  - **Double-word** = 64-bit quantity

1.1.3 Reserved, Undefined, and Unimplemented Terminology

In tables where register bit fields are defined, the following conventions are used to indicate undefined and unimplemented function. Further, types of bits and bit fields are defined using the abbreviations as shown in **Table 1-1**.

Table 1-1 Bit Function Terminology

Function of Bits	Description
Unimplemented	Register bit fields named **0** indicate unimplemented functions with the following behavior. – Reading these bit fields returns 0. – Writing these bit fields has no effect. These bit fields are reserved. When writing, software should always set such bit fields to 0 in order to preserve compatibility with future products.
Undefined	Certain bit combinations in a bit field can be labeled “Reserved”, indicating that the behavior of the TC1765 is undefined for that combination of bits. Setting the register to undefined bit combinations may lead to unpredictable results. Such bit combinations are reserved. When writing, software must always set such bit fields to legal values as given in the tables.
rw	The bit or bit field can be read and written.
r	The bit or bit field can only be read (read-only).
w	The bit or bit field can only be written (write-only).
h	The bit or bit field can also be modified by hardware (such as a status bit). This symbol can be combined with ‘rw’ or ‘r’ bits to ‘rwh’ and ‘rh’ bits.

1.1.4 Register Access Modes

Read and write access to registers and memory locations are sometimes restricted. In memory and register access tables, the following terms are used.

Table 1-2 Access Terms

Symbol	Description
U	Access permitted in User Mode 1 only. *User Mode definitions see bit PSW_IO at* ***Page 2-20****.*
SV	Access permitted in Supervisor Mode.
R	Read-only register.
32	Only 32-bit word accesses are permitted to that register/address range.
E	Endinit protected register/address.
PW	Password protected register/address.
NC	No change, indicated register is not changed.
BE	Indicates that an access to this address range generates a Bus Error.
nBE	Indicates that no Bus Error is generated when accessing this address range, even though it is either an access to an undefined address or the access does not follow the given rules.
nE	Indicates that no Error is generated when accessing this address or address range, even though the access is to an undefined address or address range. True for CPU accesses (MTCR/MFCR) to undefined addresses in the CSFR range.
X	Undefined value or bit.

1.1.5 Abbreviations

ADC	Analog-to-Digital Converter
AGPR	Address General Purpose Register
ALU	Arithmetic and Logic Unit
ASC	Asynchronous/Synchronous Serial Controller
BCU	Bus Control Unit
BROM	Boot ROM
CAN	Controller Area Network (License Bosch)
CISC	Complex Instruction Set Computing
CPS	CPU Slave Interface Registers
CPU	Central Processing Unit
CSFR	Core Special Function Registers
DGPR	Data General Purpose Register
DMU	Data Memory Unit

EBU	External Bus Unit
FPI	Flexible Peripheral Interconnect (Bus)
GPR	General Purpose Register
GPTA	General Purpose Timer Array
GPTU	General Purpose Timer Unit
ICACHE	Instruction Cache
I/O	Input / Output
NMI	Non-Maskable Interrupt
OCDS	On-Chip Debug Support
OVRAM	Code Overlay Memory
PCP	Peripheral Control Processor
PMU	Program Memory Unit
PLL	Phase Locked Loop
PCODE	PCP Code Memory
PMU	Program Memory Unit
PRAM	PCP Parameter RAM
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
RTC	Real Time Clock
SCU	System Control Unit
SDLM	Serial Data Link Module (J1850)
SFR	Special Function Register
SPRAM	Scratch-Pad Code Memory
SRAM	Static Data Memory
SSC	Synchronous Serial Controller
STM	System Timer
WDT	Watchdog Timer

1.2 System Architecture Features of the TC1765

The TC1765 combines three powerful technologies within one silicon die, achieving new levels of power, speed, and economy for embedded applications:

- Reduced Instruction Set Computing (RISC) processor architecture
- Digital signal processing (DSP) operations and addressing modes
- On-chip memories and peripherals

DSP operations and addressing modes provide the computational power necessary to efficiently analyze complex real-world signals. The RISC load/store architecture provides high computational bandwidth with low system cost. On-chip memory and peripherals are designed to support even the most demanding high-bandwidth real-time embedded control-systems tasks.

Additional high-level features of the TC1765 include:

- Program Memory Unit — instruction memory and instruction cache
- Data Memory Unit — data memory and data cache
- Serial communication interfaces — flexible synchronous and asynchronous modes
- DMA Controller — DMA operations and interrupt servicing
- General purpose timers
- On-chip debugging and emulation facilities
- Flexible interconnections to external components
- Flexible power-management

The TC1765 is a high performance microcontroller with TriCore CPU, program and data memories, buses, bus arbitration, an interrupt controller, a peripheral control processor several on-chip peripherals, and an external bus interface. The TC1765 is designed to meet the needs of the most demanding embedded control systems applications where the competing issues of price/performance, real-time responsiveness, computational power, data bandwidth, and power consumption are key design elements.

The TC1765 offers several versatile on-chip peripheral units such as serial controllers, timer units, and Analog-to-Digital converters. Within the TC1765, all these peripheral units are connected to the TriCore CPU/system via the Flexible Peripheral Interconnect (FPI) Bus. Several I/O lines on the TC1765 ports are reserved for these peripheral units to communicate with the external world.

High Performance 32-Bit CPU

- 32-bit architecture with 4 GBytes unified data, program, and input/output address space
- Fast automatic context-switch
- Multiply-accumulate unit
- Saturating integer arithmetic
- High performance on-chip peripheral bus (FPI Bus)
- Register based design with multiple variable register banks

- Bit handling
- Packed data operations
- Zero overhead loop
- Precise exceptions
- Flexible power management

Instruction Set with High Efficiency

- 16/32-bit instructions for reduced code size
- Data types include: Boolean, array of bits, character, signed and unsigned integer, integer with saturation, signed fraction, double word integers, and IEEE-754 single precision floating-point
- Data formats include: Bit, 8-bit byte, 16-bit half word, 32-bit word, and 64-bit double word data formats
- Powerful instruction set
- Flexible and efficient addressing mode for high code density

External Bus Interface

- Programmable external bus interface for low cost system implementation
- Glueless interface to a wide selection of external memories
- 8/16/32 bit data transfer
- Support for big endian byte ordering at bus interface
- Flexible address generation and access timing

Integrated On-Chip Memory

- Code memory
  - 8 KBytes Boot ROM (BROM)
  - 16 KBytes Scratch-pad RAM (SPRAM)
  - Optional 1 KByte Instruction Cache (ICACHE) for external code memory accesses
- Data memory
  - 32 KBytes data memory (SRAM) for standby operation during power-down

Interrupt System

- TBD Service Request Nodes (SRNs)
- Flexible interrupt prioritizing scheme with 256 interrupt priority levels
- Fast interrupt response
- Service requests are serviced by CPU or DMA

DMA Controller

- 2×4 independent DMA channels
- DMA requests triggered by software or hardware

- Individually programmable operating modes for each DMA Channel
  - Single Mode: stop and disable DMA Channel after a predefined number of data transactions
  - Continuous Mode: stop and remain DMA Channel enabled after a predefined number of data transactions
- 32-bit addressing mode
  - Source and destination offset programmable in steps from 0 to 255 Bytes
  - Circular buffer addressing mode supported
- Programmable data width of 8-Bit, 16-Bit, 32-Bit per data transaction
- Register set for each DMA channel:
  - Source and destination start address register
  - Source and destination end address register
  - Channel control and status register
- Bus bandwidth allocation
- Flexible interrupt generation

I/O Ports With Individual Bit Addressability

- Push/pull or open drain output mode
- Selectable input thresholds
- Programmable output speed and characteristic

Package

- L-BGA-260 package, 1 mm pitch
  - TC1765N without OCDS-2 trace port
  - TC1765T with OCDS-2 trace port for extended debugging

Temperature Ranges

- Ambient temperature: -40 °C to +125 °C
- Max. junction temperature: +150 °C

System Clock Frequency

- Maximum System Clock Frequency: 40 MHz

Complete Development Support

A variety of software and hardware development tools for the 32-bit microcontroller TC1765 is available from experienced international tool suppliers. The development environment for the Infineon 32-bit microcontroller includes the following tools:

- TC1765T with OCDS-2 trace port for extended debugging
- Embedded Development Environment for TriCore Products

- The TC1765 On-chip Debug Support (OCDS) provides a JTAG port for communication between external hardware and the system.
- The Flexible Peripheral Interconnect Bus (FPI Bus) for on-chip interconnections and the FPI Bus control unit (BCU).
- The System Timer (STM) with high-precision, long-range timing capabilities.
- The TC1765 includes a power management system, a watchdog timer as well as a reset logic.

1.3 Block Diagram

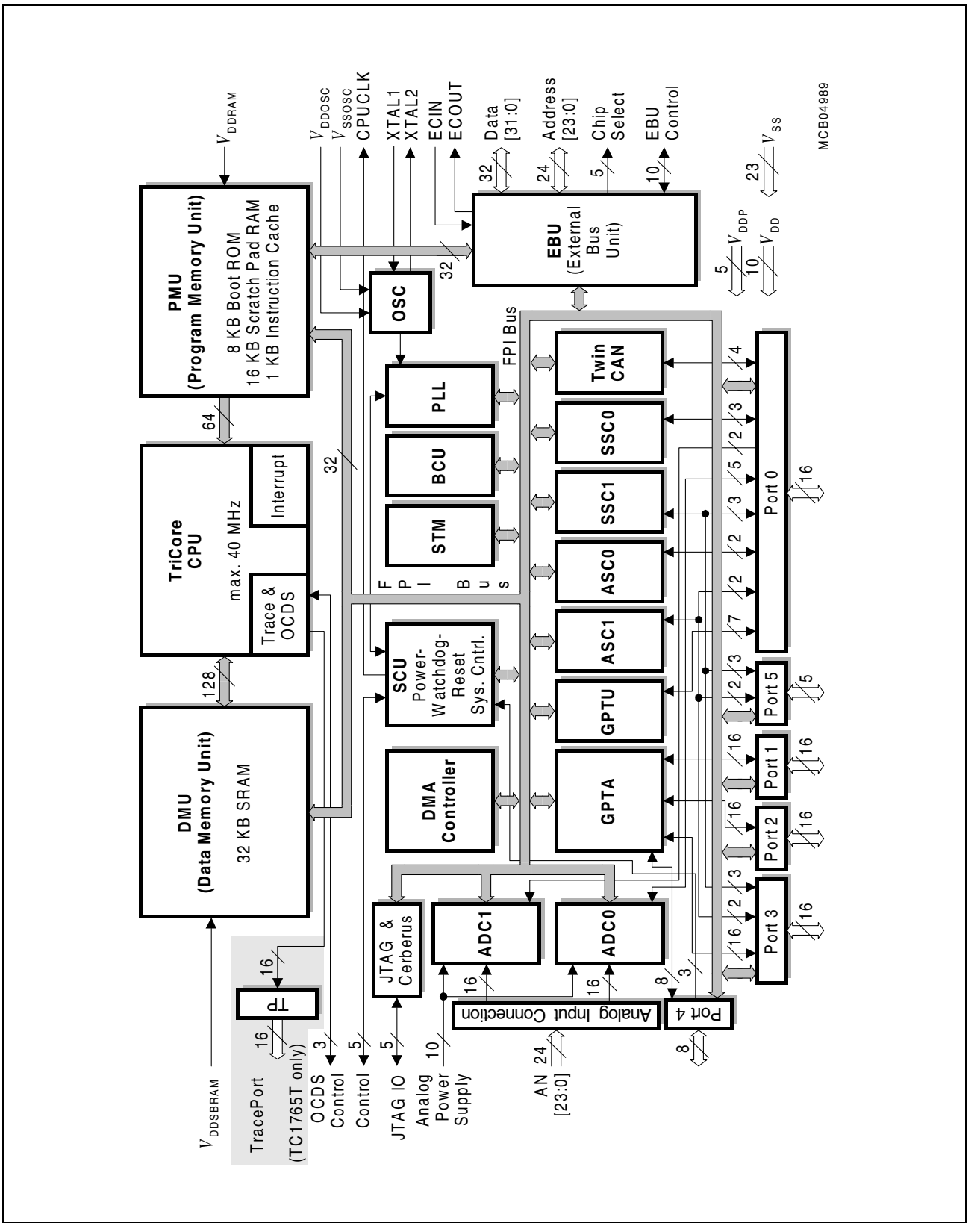


Figure 1-1 TC1765 Block Diagram

1.4 On-Chip Peripheral Units of the TC1765

The following peripherals are all described in detail in the chapters of this “TC1765 Peripheral Units’ User’s Manual”:

- Two Asynchronous/Synchronous Serial Interfaces with baud rate generator, parity, framing, and overrun error detection
- Two High Speed Synchronous Serial Interfaces with programmable data length and shift direction
- TwinCAN Controller with two interconnected CAN nodes for high efficiency data handling via FIFO buffering and gateway data transfer
- Multifunctional General Purpose Timer Unit with three 32-bit timer/counter
- General Purpose Timer Array with a powerful set of digital signal filtering and timer functionality to realize autonomous and complex Input/Output management
- Two Analog-to-Digital Converter Units with 8-bit, 10-bit, or 12-bit resolution and sixteen analog inputs each

The next sections within this chapter provide an overview of these peripheral units.

Note: Additionally to the “TC1765 System Units User’s Manual”, a 2nd document, the “TC1765 Peripheral Units User’s Manual”, is available. These two User’s Manuals together with the “TriCore Architecture Manual” are required for the understanding the complete TC1765 microcontroller functionality.

1.4.1 Serial Interfaces

The TC1765 includes five serial peripheral interface units:

- Two Asynchronous/Synchronous Serial Interfaces (ASC0 and ASC1)
- Two High-Speed Synchronous Serial Interfaces (SSC0 and SSC1)
- One TwinCAN Interface

1.4.1.1 Asynchronous/Synchronous Serial Interfaces

Figure 1-2 shows a global view of the functional blocks of the two Asynchronous/ Synchronous Serial interfaces.

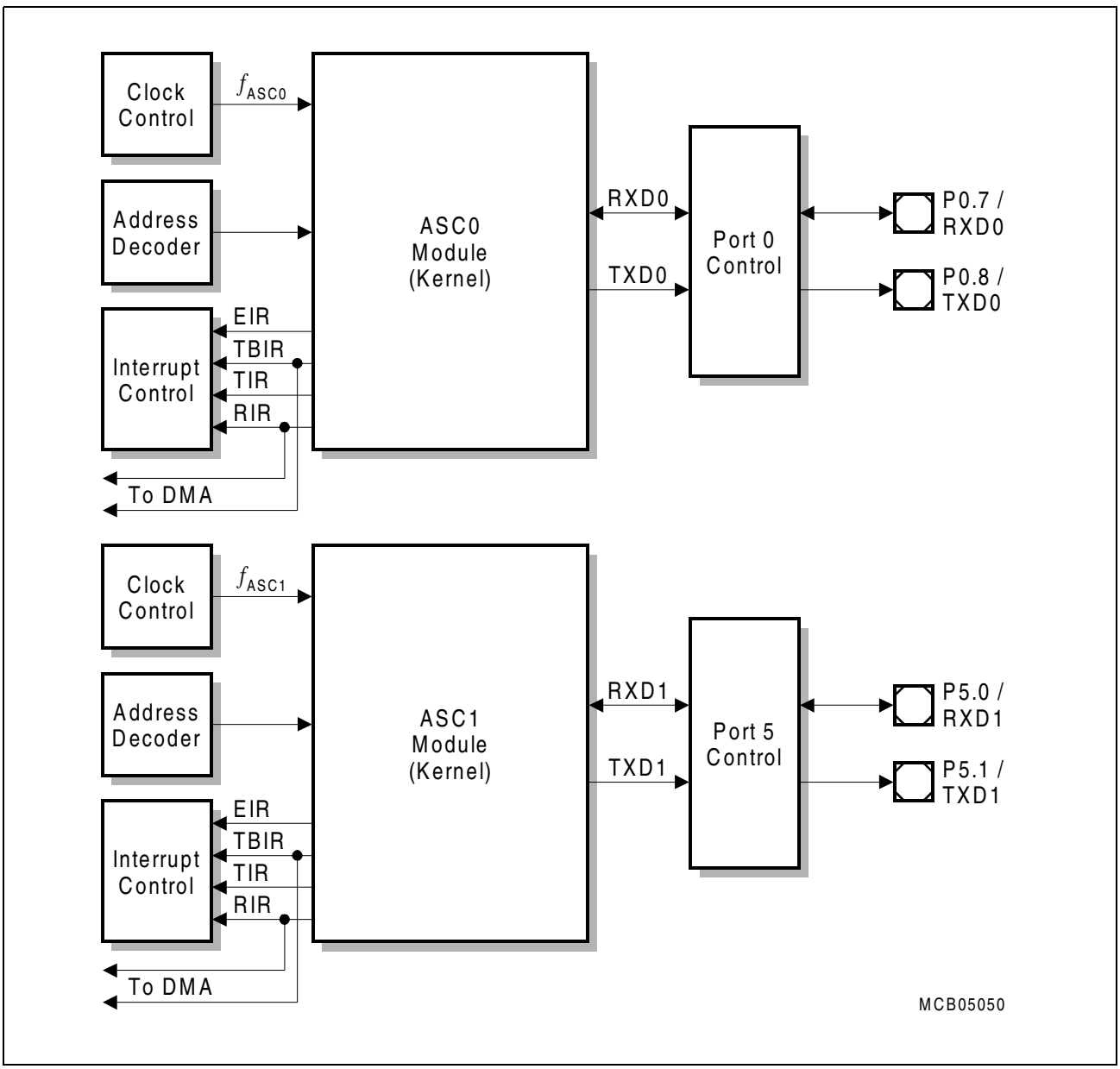


Figure 1-2 General Block Diagram of the ASC Interfaces

The Asynchronous/Synchronous Serial Interfaces provide serial communication between the TC1765 and other microcontrollers, microprocessors, or external peripherals.

The ASC supports full-duplex asynchronous communication and half-duplex synchronous communication. In Synchronous Mode, data is transmitted or received synchronous to a shift clock which is generated by the ASC internally. In Asynchronous Mode, 8-bit or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection are provided to increase the reliability of data transfers. Transmission and reception of data are double-buffered. For multiprocessor communication, a mechanism is included to distinguish address bytes from data bytes. Testing is supported by a loop-back option. A 13-bit baud rate generator provides the ASC with a separate serial clock signal that can be very accurately adjusted by a prescaler implemented as a fractional divider.

Each ASC module, ASC0 and ASC1, communicates with the external world via two I/O lines. The RXD line is the receive data input signal (in Synchronous Mode also output). TXD is the transmit output signal. Clock control, address decoding, and interrupt service request control are managed outside the ASC module kernel.

Features:

- Full-duplex asynchronous operating modes
  - 8-bit or 9-bit data frames, LSB first
  - Parity bit generation/checking
  - One or two stop bits
  - Baud rate from 2.5 MBaud to 0.6 Baud (@ 40 MHz clock)
  - Multiprocessor mode for automatic address/data byte detection
  - Loop-back capability
- Half-duplex 8-bit synchronous operating mode
  - Baud rate from 5 MBaud to 406.9 Baud (@ 40 MHz clock)
- Double buffered transmitter/receiver
- Interrupt generation
  - On a transmit buffer empty condition
  - On a transmit last bit of a frame condition
  - On a receive buffer full condition
  - On an error condition (frame, parity, overrun error)

1.4.1.2 High-Speed Synchronous Serial Interfaces

Figure 1-3 shows a global view of the functional blocks of the two High-Speed Synchronous Serial interfaces.

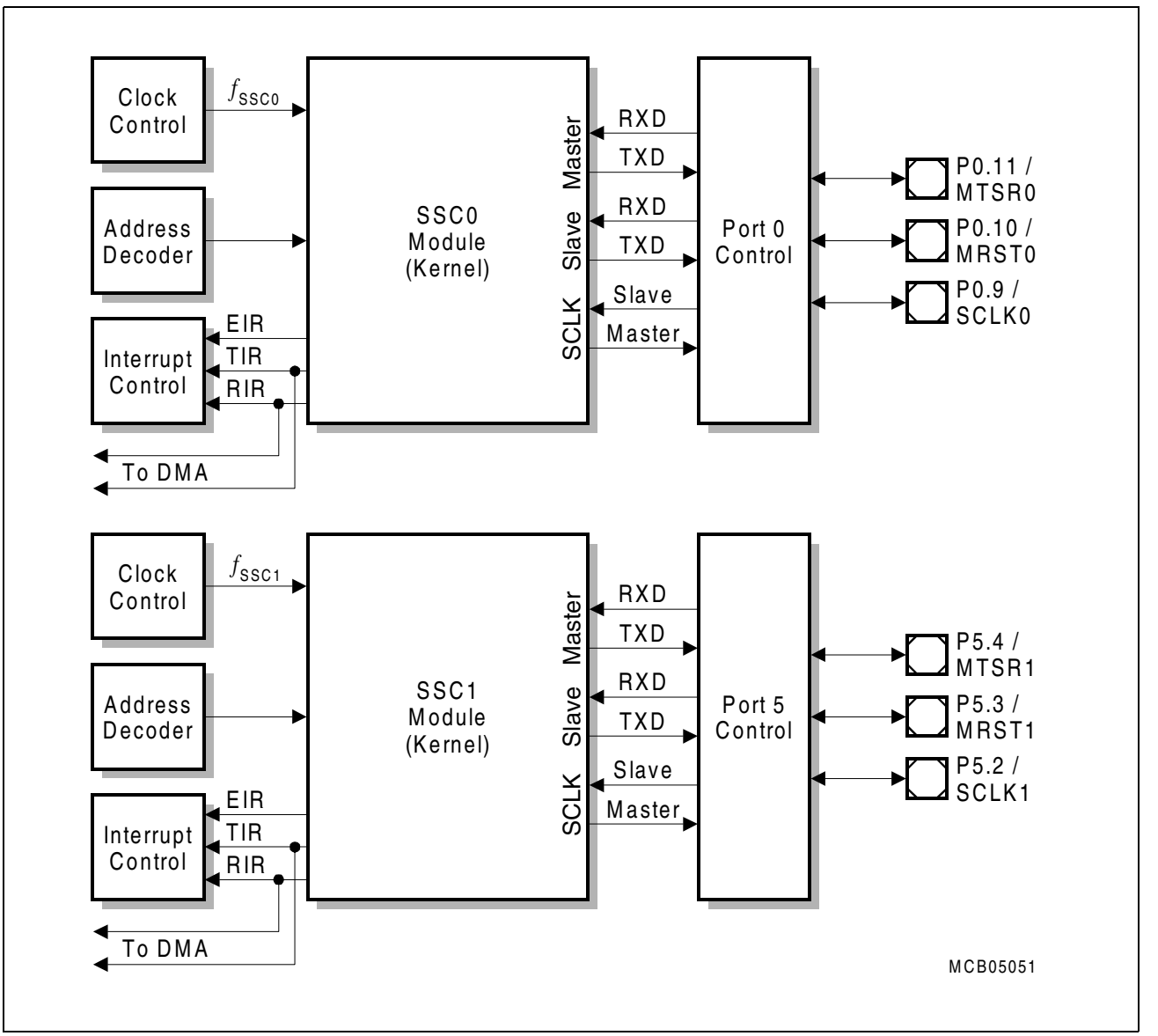


Figure 1-3 General Block Diagram of the SSC Interfaces

The SSC supports full-duplex and half-duplex serial synchronous communication up to 20 MBaud (@ 40 MHz module clock) with receive and transmit FIFO support. The serial clock signal can be generated by the SSC itself (master mode) or can be received from an external master (slave mode). Data width, shift direction, clock polarity, and phase are programmable. This allows communication with SPI-compatible devices. Transmission and reception of data are double-buffered. A 16-bit baud rate generator provides the SSC with a separate serial clock signal.

Each of the SSC modules has three I/O lines, located at Port 0 and Port 5. Each of the SSC modules is further supplied by separate clock control, interrupt control, address decoding, and port control logic.

Features:

- Master and slave mode operation
  - Full-duplex or half-duplex operation
- Flexible data format
  - Programmable number of data bits: 2-bit to 16-bit
  - Programmable shift direction: LSB or MSB shift first
  - Programmable clock polarity: idle low or high state for the shift clock
  - Programmable clock/data phase: data shift with leading or trailing edge of the shift clock
- Baud rate generation from 20 MBaud to 305.18 Baud (@ 40 MHz module clock)
- Interrupt generation
  - On a transmitter empty condition
  - On a receiver full condition
  - On an error condition (receive, phase, baud rate, transmit error)
- Three-pin interface
  - Flexible SSC pin configuration
- 8-stage receive FIFO (RXFIFO) and 8-stage transmit FIFO (TXFIFO)
  - Independent control of RXFIFO and TXFIFO
  - 2 to 16-bit FIFO data width
  - Programmable receive/transmit interrupt trigger level
  - Receive and transmit FIFO filling level indication
  - Overrun error generation
  - Underflow error generation

1.4.1.3 TwinCAN Interface

The TwinCAN module contains two Full-CAN nodes operating independently or exchanging data and remote frames via a gateway function. Transmission and reception of CAN frames are handled in accordance to CAN specification V2.0 part B (active). Each of the two Full-CAN nodes can receive and transmit standard frames with 11-bit identifiers as well as with extended frames with 29-bit identifiers.

Both CAN nodes share the TwinCAN module's resources to optimize the CAN bus traffic handling and to minimize the CPU load. The flexible combination of Full-CAN functionality and FIFO architecture reduces the efforts to fulfill the real-time requirements of complex embedded control applications. Improved CAN bus monitoring functionality as well as the increased number of message objects permit precise and convenient CAN bus traffic handling.

Depending on the application, each of the 32 message objects can be individually assigned to one of the two CAN nodes. Gateway functionality allows automatic data exchange between two separate CAN bus systems, which decreases CPU load and improves the real time behavior of the entire system.

The bit timings for both CAN nodes are derived from the peripheral clock (f_{CAN}) and are programmable up to a data rate of 1 MBaud. A pair of receive and transmit pins connect each CAN node to a bus transceiver.

The TwinCAN module has four I/O lines located at Port 0. The TwinCAN module is further supplied by a clock control, interrupt control, address decoding, and port control logic.

Features:

- CAN functionality conforms to CAN specification V2.0 B active.
- Dedicated control registers are provided for each CAN node.
- A data transfer rate up to 1 MBaud is supported.
- Flexible and powerful message transfer control and error handling capabilities are implemented.
- Full-CAN functionality: 32 message objects can be individually
  - Assigned to one of the two CAN nodes
  - Configured as transmit or receive object
  - Participate in a 2, 4, 8, 16 or 32 message buffer with FIFO algorithm
  - Set up to handle frames with 11-bit or 29-bit identifiers
  - Provided with programmable acceptance mask register for filtering
  - Monitored via a frame counter
  - Configured to Remote Monitoring Mode
- Up to eight individually programmable interrupt nodes can be used.
- CAN Analyzer Mode for bus monitoring is implemented.

Figure 1-4 shows the functional units of the TwinCAN module.

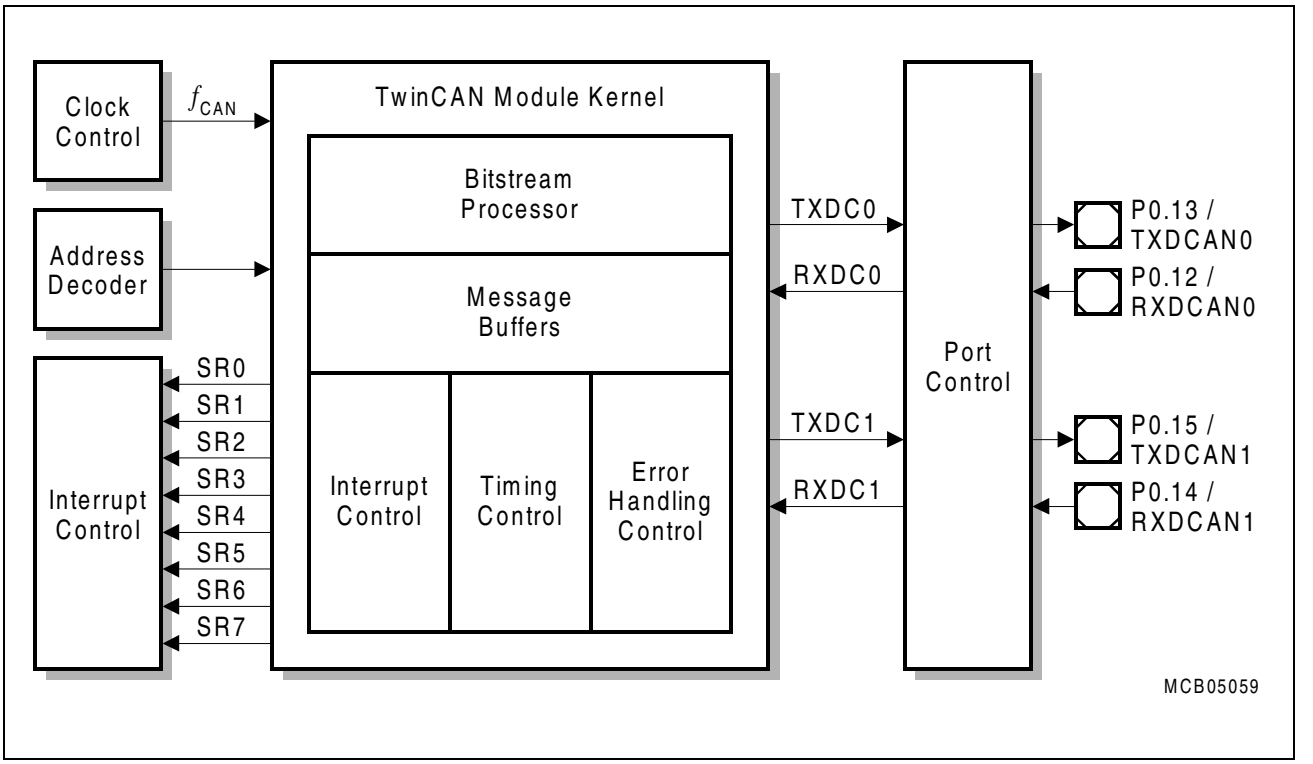


Figure 1-4 General Block Diagram of the TwinCAN Interfaces

1.4.2 Timer Units

The TC1765 includes two timer units:

- General Purpose Timer Unit (GPTU)
- General Purpose Timer Array (GPTA)

1.4.2.1 General Purpose Timer Unit

Figure 1-5 shows a global view of all functional blocks of the General Purpose Timer Unit (GPTU) module.

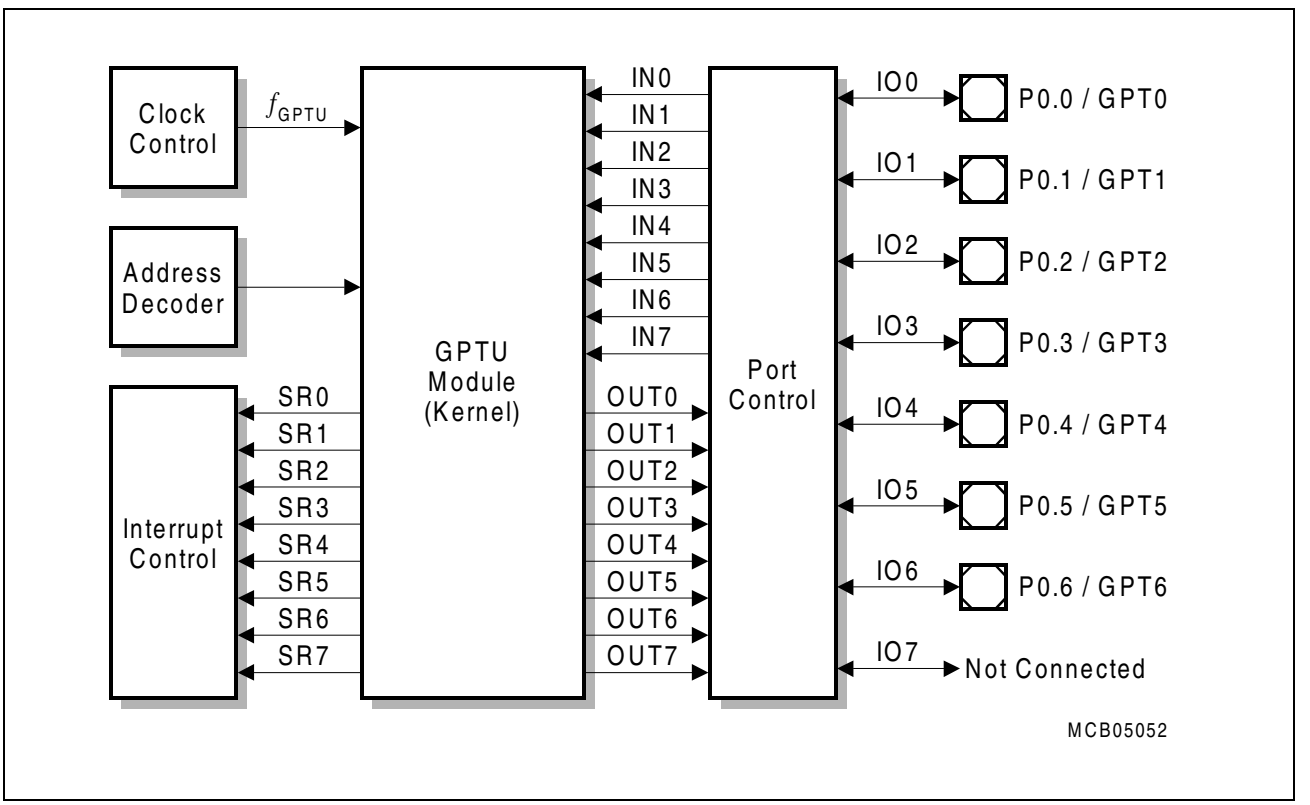


Figure 1-5 General Block Diagram of the GPTU Interface

The GPTU consists of three 32-bit timers designed to solve such application tasks as event timing, event counting, and event recording. The GPTU communicates with the external world via eight inputs and eight outputs located at Port 0.

The I/O has three timers (T0, T1, and T2) can operate independently from each other, or can be combined.

General Features:

- All timers are 32-bit precision timers with a maximum input frequency of f_{GPTU}
- Events generated in T0 or T1 can be used to trigger actions in T2
- Timer overflow or underflow in T2 can be used to clock either T0 or T1
- T0 and T1 can be concatenated to form one 64-bit timer

Features of T0 and T1:

- Each timer has a dedicated 32-bit reload register with automatic reload on overflow
- Timers can be split into individual 8-, 16-, or 24-bit timers with individual reload registers
- Overflow signals can be selected to generate service requests, pin output signals, and T2 trigger events
- Two input pins can define a count option

Features of T2:

- Count up or down is selectable
- Operating modes:
  - Timer
  - Counter
  - Quadrature counter (incremental/phase encoded counter interface)
- Options:
  - External start/stop, one-shot operation, timer clear on external event
  - Count direction control through software or an external event
  - Two 32-bit reload/capture registers
- Reload modes:
  - Reload on overflow or underflow
  - Reload on external event: positive transition, negative transition, or both transitions
- Capture modes:
  - Capture on external event: positive transition, negative transition, or both transitions
  - Capture and clear timer on external event: positive transition, negative transition, or both transitions
- Can be split into two 16-bit counter/timers
- Timer count, reload, capture, and trigger functions can be assigned to input pins. T0 and T1 overflow events can also be assigned to these functions
- Overflow and underflow signals can be used to trigger T0 and/or T1 and to toggle output pins
- T2 events are freely assignable to the service request nodes

1.4.2.2 General Purpose Timer Array

The General Purpose Timer Array (GPTA) provides important digital signal filtering and timer support whose combination enables autonomous and complex functionalities. This architecture allows easy implementation and easy validation of any kind of timer functions.

Figure 1-6 shows a global block diagram of the General Purpose Timer Array (GPTA).

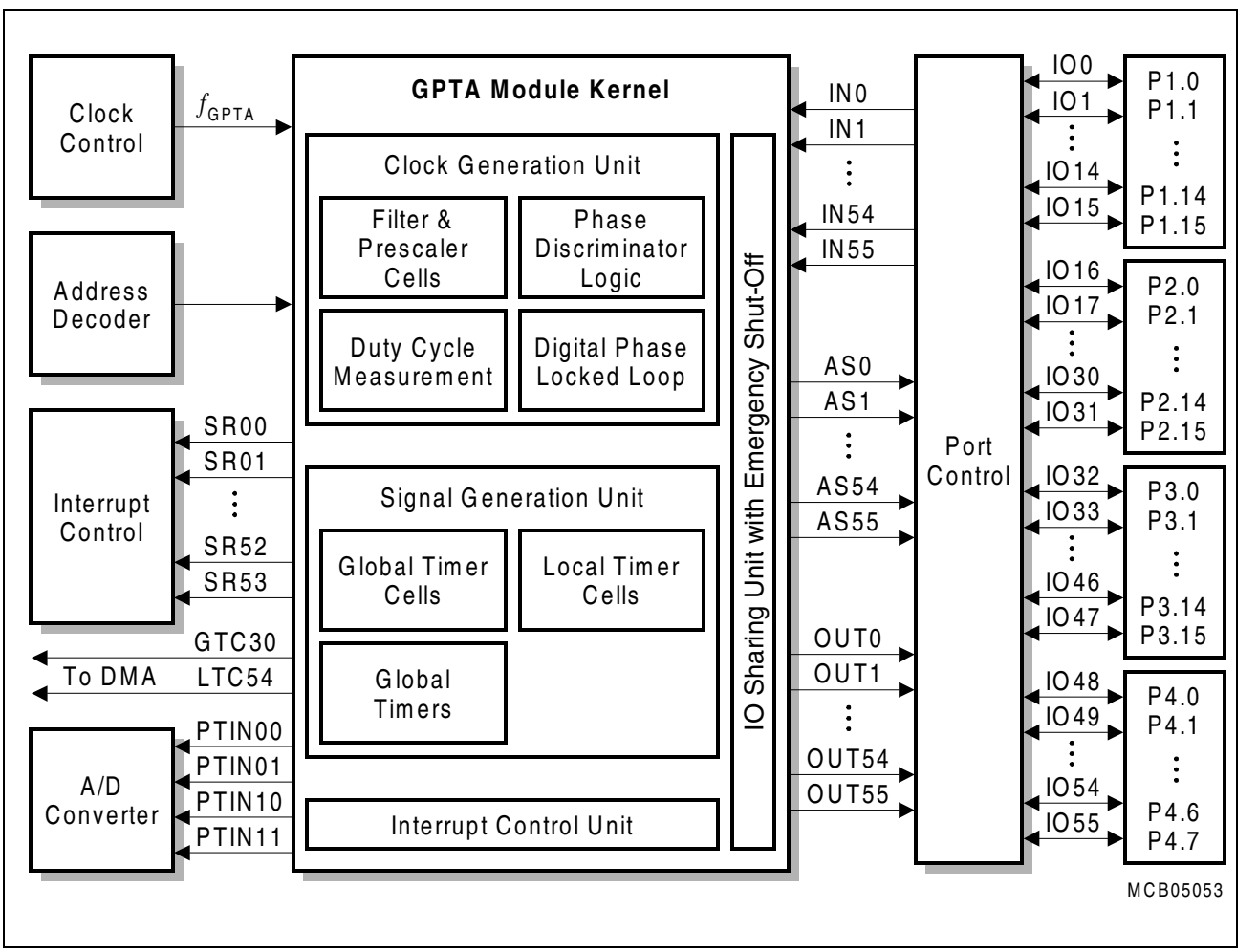


Figure 1-6 GPTA Module Block Diagram

The GPTA module has 56 input signals and 56 output signals which are connected with 56 Port 1, Port 2, Port 3, and Port 4 pins.

The General Purpose Timer Array (GPTA) provides a set of hardware modules required for high speed digital signal processing:

- Filter and Prescaler Cells (FPC) support input noise filtering and prescaler operation.
- Phase Discrimination Logic units (PDL) decode the direction information output by a rotation tracking system.
- Duty Cycle Measurement Cells (DCM) provide pulse width measurement capabilities.
- A Digital Phase Locked Loop unit (PLL) generates a programmable number of GPTA module clock ticks during an input signal's period.
- Global Timer units (GT) driven by various clock sources are implemented to operate as a time base for the associated "Global Timer Cells".
- Global Timer Cells (GTC) can be programmed to capture the contents of a Global Timer on an event that occurred at an external port pin or at an internal FPC output. A GTC may be also used to control an external port pin with the result of an internal compare operation. GTCs can be logically concatenated to provide a common external port pin with a complex signal waveform.
- Local Timer Cells (LTC) operating in Timer, Capture, or Compare Mode may be also logically tied together to drive a common external port pin with a complex signal waveform. LTCs — enabled in Timer Mode or Capture Mode — can be clocked or triggered by
  - A prescaled GPTA module clock,
  - An FPC, PDL, DCM, PLL, or GTC output signal line,
  - An external port pin.

Some input lines driven by processor I/O pads may be shared by a LTC and a GTC cell to trigger their programmed operation simultaneously.

The following list summarizes all blocks supported:

Clock Generation Unit:

- Filter and Prescaler Cell (FPC):
  - Six independent units.
  - Three operating modes (Prescaler, Delayed Debounce Filter, Immediate Debounce Filter).
  - f_{GPTA} down-scaling capability.
  - $f_{GPTA}/2$ maximum input signal frequency in Filter Mode.
- Phase Discriminator Logic (PDL):
  - Two independent units.
  - Two operating modes (2 and 3 sensor signals).
  - $f_{GPTA}/4$ maximum input signal frequency in 2-sensor mode, $f_{GPTA}/6$ maximum input signal frequency in 3-sensor mode.
- Duty Cycle Measurement (DCM):
  - Four independent units.
  - 0 - 100% margin and time-out handling.

  - f_{GPTA} maximum resolution.
  - f_{GPTA}/2 maximum input signal frequency.
- Digital Phase Locked Loop (PLL):
  - One unit.
  - Arbitrary multiplication factor between 1 and 65535.
  - f_{GPTA} maximum resolution.
  - f_{GPTA}/2 maximum input signal frequency.

Signal Generation Unit:

- Global Timers (GT):
  - Two independent units.
  - Two operating modes (Free Running Timer and Reload Timer).
  - 24-bit data width.
  - f_{GPTA} maximum resolution.
  - f_{GPTA}/2 maximum input signal frequency.
- Global Timer Cell (GTC):
  - 32 independent units.
  - Two operating modes (Capture, Compare and Capture after Compare).
  - 24-bit data width.
  - f_{GPTA} maximum resolution.
  - f_{GPTA}/2 maximum input signal frequency.
- Local Timer Cell (LTC):
  - 64 independent units.
  - Three operating modes (Timer, Capture and Compare).
  - 16-bit data width.
  - f_{GPTA} maximum resolution.
  - f_{GPTA}/2 maximum input signal frequency.

Interrupt Control Unit:

- 111 interrupt sources generating 54 service requests.

I/O Sharing Unit:

- Interconnecting input and output lines from FPC, GTC, LTC and ports.
- Emergency function.

1.4.3 Analog-to-Digital Converters

The two on-chip ADC modules of the TC1765 are analog to digital converters with 8-bit, 10-bit or 12-bit resolution including sample & hold functionality. The A/D converters operate by the method of the successive approximation. A multiplexer selects between up to 16 analog input channels for each ADC module. The 24 analog inputs are switched to the analog input channels of the ADC modules by a fixed scheme. Conversion requests are generated either under software control or by hardware (GPTA). An automatic self-calibration adjusts the ADC modules to changing temperatures or process variations.

Features

- 8-bit, 10-bit, 12-bit A/D Conversion
- Successive approximation conversion method
- Total Unadjusted Error (TUE) of ±2 LSB @ 10-bit resolution
- Integrated sample and hold functionality
- 24 analog input pins / 16 analog input channels of each ADC module
- Fix assignment of 24 analog input pins to the 32 ADC0/ADC1 input channels
- Dedicated control and status registers for each analog channel
- Flexible conversion request mechanisms
- Selectable reference voltages for each channel
- Programmable sample and conversion timing schemes
- Limit checking
- Flexible ADC module service request control unit
- Synchronization of the two on-chip A/D Converters
- Automatic control of an external analog input multiplexer for ADC0
- Equidistant samples initiated by timer
- Two trigger inputs, connected with the General Purpose Timer Array (GPTA)
- Two external trigger input pins of each ADC for generating conversion requests
- Power reduction and clock control feature

Figure 1-7 shows a global view of the ADC module kernel with the module specific interface connections.

The ADC module communicates with the external world via five digital I/O lines and sixteen analog inputs. Clock control, address decoding, digital I/O port control, and service request generation is managed outside the ADC module kernel. The end of a conversion is indicated for each channel n (n = 0-15) by a pulse on the output signals SRCHn. These signals can be used to trigger a DMA transfer to read the conversion result automatically. Two trigger inputs and a synchronization bridge are used for internal control purposes.

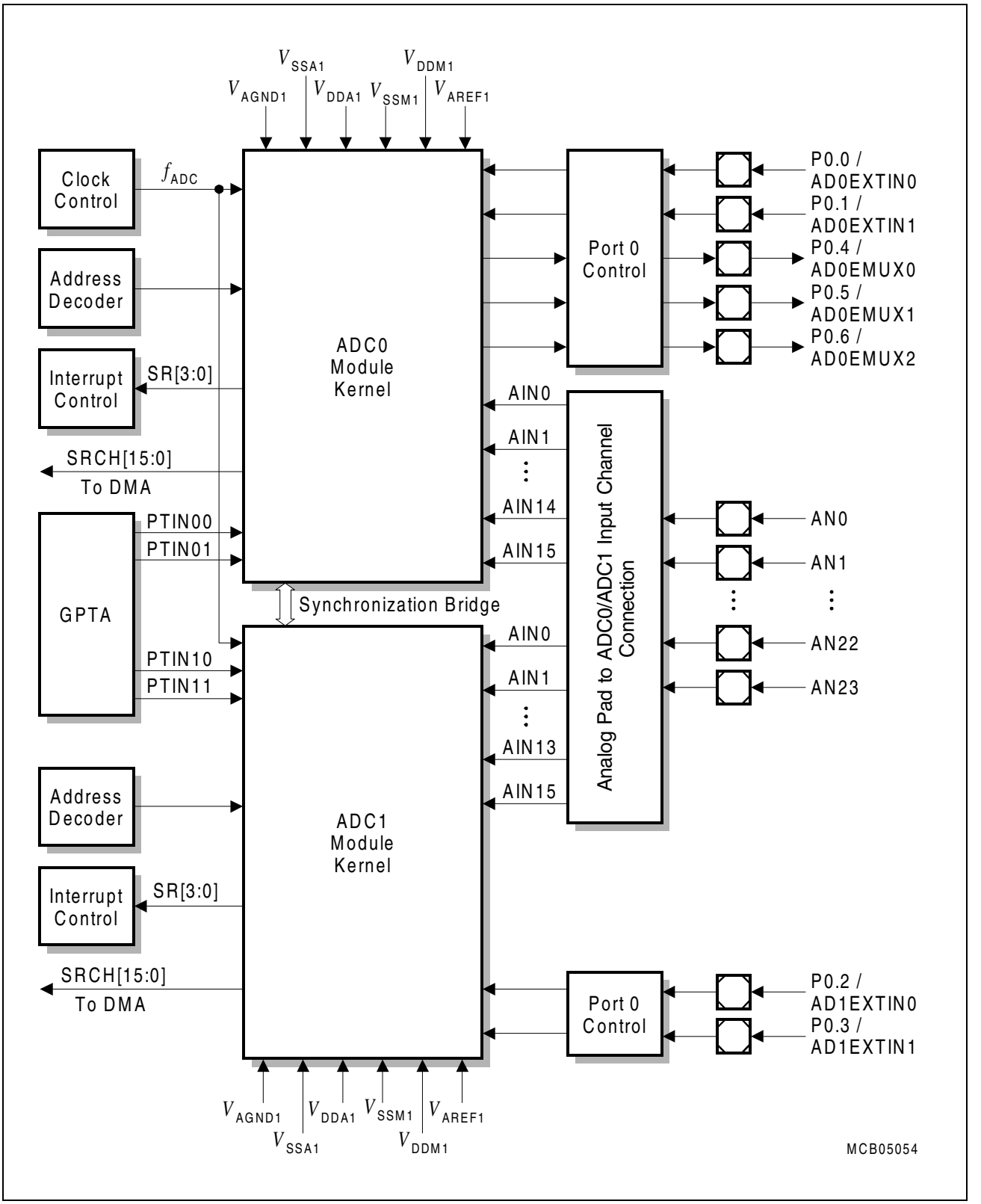


Figure 1-7 General Block Diagram of the ADC Interface

1.5 Pin Definitions and Functions

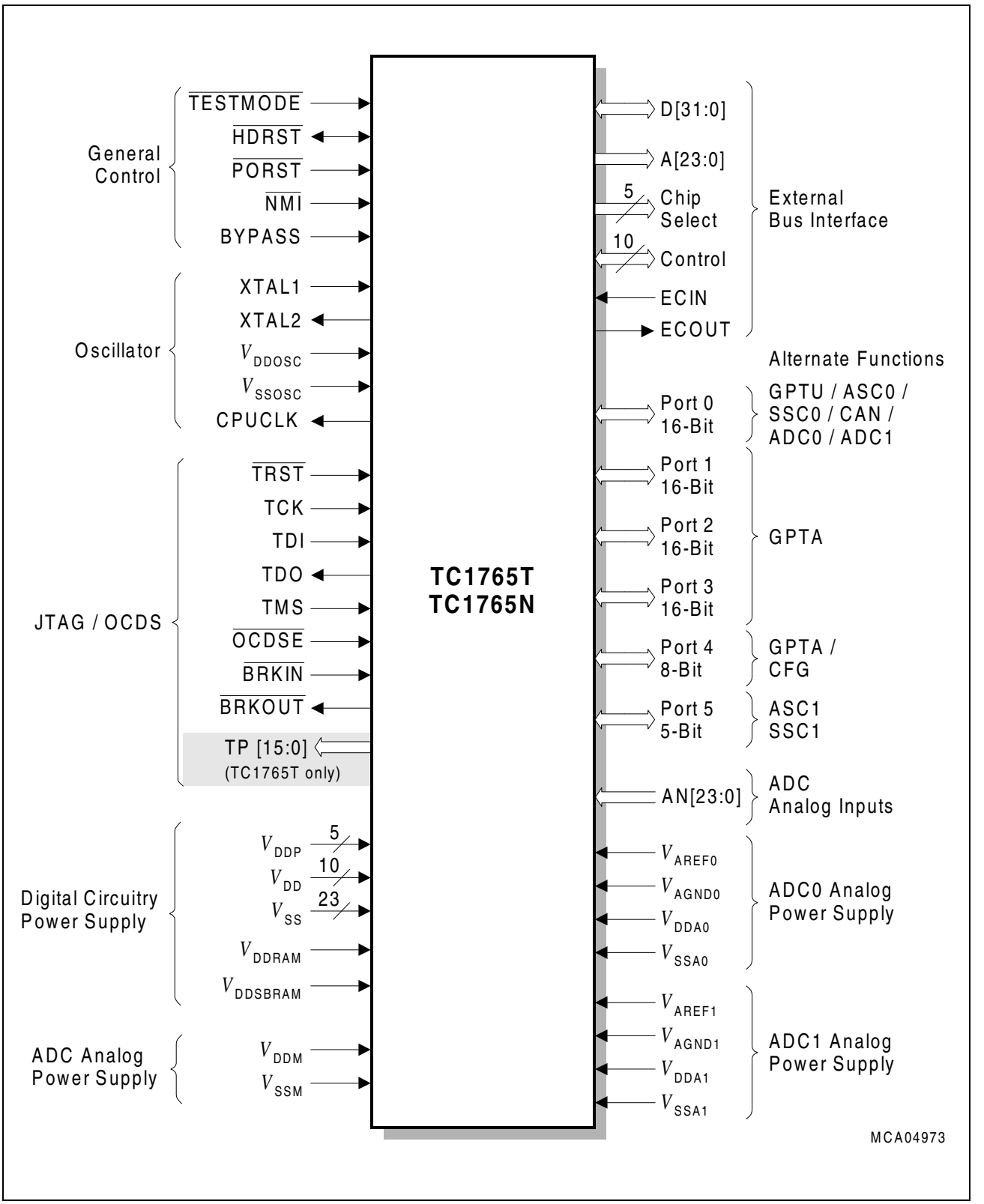


Figure 1-8 TC1765 Pin Configuration

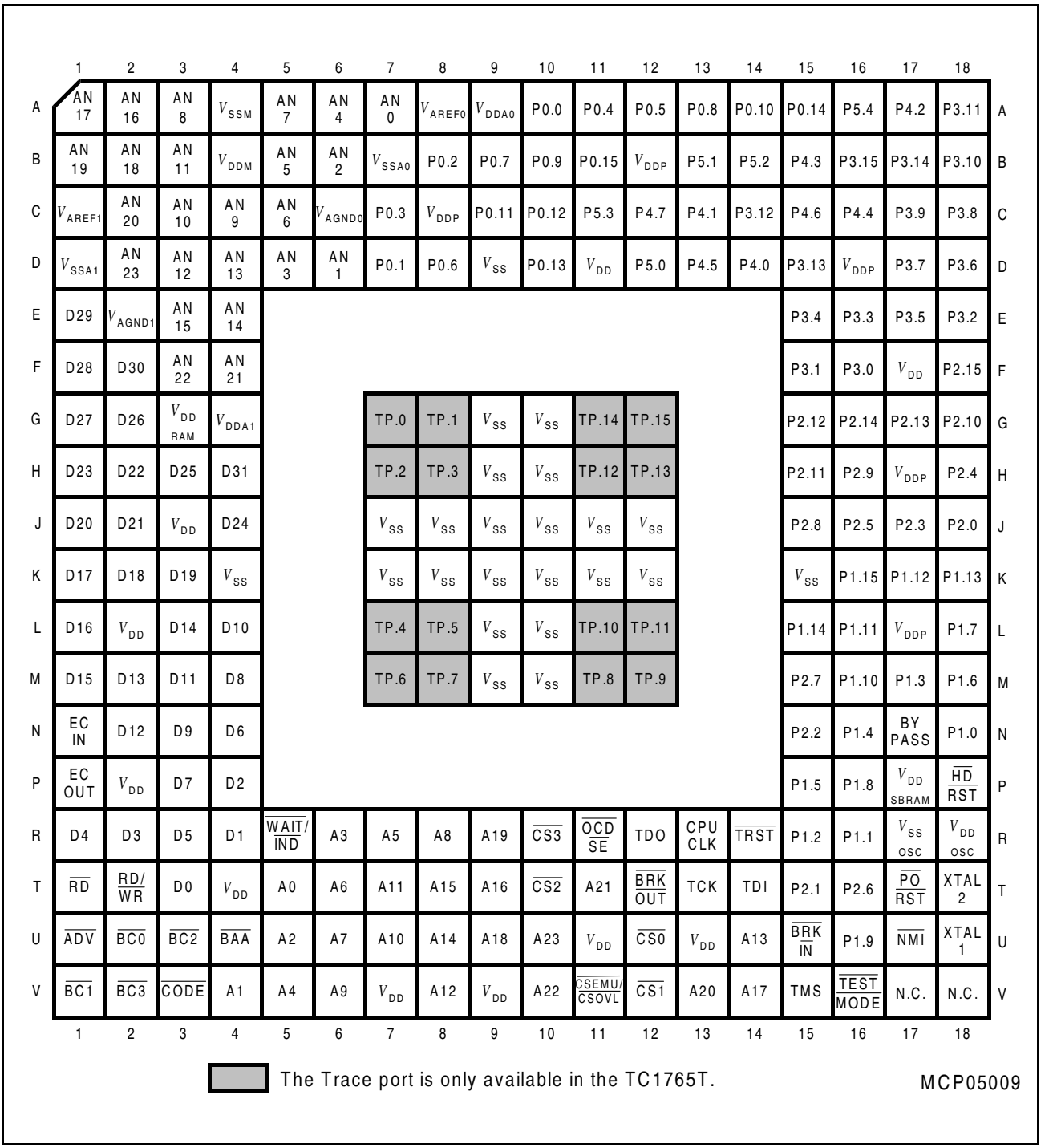

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
A AN17 AN16 AN8 V_{SSM} AN7 AN4 AN0 V_{AREF0} V_{DDA0} P0.0 P0.4 P0.5 P0.8 P0.10 P0.14 P5.4 P4.2 P3.11 A
B AN19 AN18 AN11 V_{DDM} AN5 AN2 V_{SSA0} P0.2 P0.7 P0.9 P0.15 V_{DDP} P5.1 P5.2 P4.3 P3.15 P3.14 P3.10 B
C V_{AREF1} AN20 AN10 AN9 AN6 V_{AGND0} P0.3 V_{DDP} P0.11 P0.12 P5.3 P4.7 P4.1 P3.12 P4.6 P4.4 P3.9 P3.8 C
D V_{SSA1} AN23 AN12 AN13 AN3 AN1 P0.1 P0.6 V_{SS} P0.13 V_{DD} P5.0 P4.5 P4.0 P3.13 V_{DDP} P3.7 P3.6 D
E D29 V_{AGND1} AN15 AN14 P3.4 P3.3 P3.5 P3.2 E
F D28 D30 AN22 AN21 P3.1 P3.0 V_{DD} P2.15 F
G D27 D26 V_{DDRAM} V_{DDA1} TP.0 TP.1 V_{SS} V_{SS} TP.14 TP.15 P2.12 P2.14 P2.13 P2.10 G
H D23 D22 D25 D31 TP.2 TP.3 V_{SS} V_{SS} TP.12 TP.13 P2.11 P2.9 V_{DDP} P2.4 H
J D20 D21 V_{DD} D24 V_{SS} V_{SS} V_{SS} V_{SS} V_{SS} V_{SS} P2.8 P2.5 P2.3 P2.0 J
K D17 D18 D19 V_{SS} V_{SS} V_{SS} V_{SS} V_{SS} V_{SS} V_{SS} V_{SS} P1.15 P1.12 P1.13 K
L D16 V_{DD} D14 D10 TP.4 TP.5 V_{SS} V_{SS} TP.10 TP.11 P1.14 P1.11 V_{DDP} P1.7 L
M D15 D13 D11 D8 TP.6 TP.7 V_{SS} V_{SS} TP.8 TP.9 P2.7 P1.10 P1.3 P1.6 M
N EC IN D12 D9 D6 P2.2 P1.4 BY PASS P1.0 N
P EC OUT V_{DD} D7 D2 P1.5 P1.8 $V_{DDSBRAM}$ $\overline{HD}$ $\overline{RST}$ P
R D4 D3 D5 D1 $\overline{WAIT}$/$\overline{IND}$ A3 A5 A8 A19 $\overline{CS3}$ $\overline{OCD}$ $\overline{SE}$ TDO CPU CLK $\overline{TRST}$ P1.2 P1.1 V_{SSOSC} V_{DDOSC} R
T $\overline{RD}$ $\overline{RD}$/$\overline{WR}$ D0 V_{DD} A0 A6 A11 A15 A16 $\overline{CS2}$ A21 $\overline{BRK}$ $\overline{OUT}$ TCK TDI P2.1 P2.6 $\overline{PO}$ $\overline{RST}$ XTAL 2 T
U $\overline{ADV}$ $\overline{BC0}$ $\overline{BC2}$ $\overline{BAA}$ A2 A7 A10 A14 A18 A23 V_{DD} $\overline{CS0}$ V_{DD} A13 $\overline{BRK}$ $\overline{IN}$ P1.9 $\overline{NMI}$ XTAL 1 U
V $\overline{BC1}$ $\overline{BC3}$ $\overline{CODE}$ A1 A4 A9 V_{DD} A12 V_{DD} A22 $\overline{CSEMU}$/$\overline{CSOVL}$ $\overline{CS1}$ A20 A17 TMS $\overline{TEST}$ $\overline{MODE}$ N.C. N.C. V
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
The Trace port is only available in the TC1765T.
MCP05009

Figure 1-9 TC1765 Pinning: L-BGA-260 Package (top view)

Table 1-3 Pin Definitions and Functions

Symbol	Pin	In Out	Functions
D[31:0]			**EBU Data Bus Lines[1)2)]** The EBU Data Bus Lines D[31:0] serve as external data bus.
D0	T3	I/O	Data bus line 0
D1	R4	I/O	Data bus line 1
D2	P4	I/O	Data bus line 2
D3	R2	I/O	Data bus line 3
D4	R1	I/O	Data bus line 4
D5	R3	I/O	Data bus line 5
D6	N4	I/O	Data bus line 6
D7	P3	I/O	Data bus line 7
D8	M4	I/O	Data bus line 8
D9	N3	I/O	Data bus line 9
D10	L4	I/O	Data bus line 10
D11	M3	I/O	Data bus line 11
D12	N2	I/O	Data bus line 12
D13	M2	I/O	Data bus line 13
D14	L3	I/O	Data bus line 14
D15	M1	I/O	Data bus line 15
D16	L1	I/O	Data bus line 16
D17	K1	I/O	Data bus line 17
D18	K2	I/O	Data bus line 18
D19	K3	I/O	Data bus line 19
D20	J1	I/O	Data bus line 20
D21	J2	I/O	Data bus line 21
D22	H2	I/O	Data bus line 22
D23	H1	I/O	Data bus line 23
D24	J4	I/O	Data bus line 24
D25	H3	I/O	Data bus line 25
D26	G2	I/O	Data bus line 26
D27	G1	I/O	Data bus line 27
D28	F1	I/O	Data bus line 28
D29	E1	I/O	Data bus line 29
D30	F2	I/O	Data bus line 30
D31	H4	I/O	Data bus line 31

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pin	In Out	Functions
A[23:0]			**EBU Address Bus Lines[3)4)]** The EBU Address Bus Lines A[23:0] serve as address bus.
A0	T5	O	Address bus line 0
A1	V4	O	Address bus line 1
A2	U5	O	Address bus line 2
A3	R6	O	Address bus line 3
A4	V5	O	Address bus line 4
A5	R7	O	Address bus line 5
A6	T6	O	Address bus line 6
A7	U6	O	Address bus line 7
A8	R8	O	Address bus line 8
A9	V6	O	Address bus line 9
A10	U7	O	Address bus line 10
A11	T7	O	Address bus line 11
A12	V8	O	Address bus line 12
A13	U14	O	Address bus line 13
A14	U8	O	Address bus line 14
A15	T8	O	Address bus line 15
A16	T9	O	Address bus line 16
A17	V14	O	Address bus line 17
A18	U9	O	Address bus line 18
A19	R9	O	Address bus line 19
A20	V13	O	Address bus line 20
A21	T11	O	Address bus line 21
A22	V10	O	Address bus line 22
A23	U10	O	Address bus line 23
			Chip Select Lines[3)5)]
$\overline{\text{CS0}}$	U12	O	Chip select output line 0
$\overline{\text{CS1}}$	V12	O	Chip select output line 1
$\overline{\text{CS2}}$	T10	O	Chip select output line 2
$\overline{\text{CS3}}$	R10	O	Chip select output line 3
$\overline{\text{CSEMU}}$/ $\overline{\text{CSOVL}}$	V11	O	**Chip Select for Emulator Region / Chip select for Emulator Overlay Memory[3)5)]**

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pin	In Out	Functions
			EBU Control Lines[1)5)] The EBU control lines are required for controlling external memory or peripheral devices.
$\overline{BC0}$	U2	O	Byte control line 0
$\overline{BC1}$	V1	O	Byte control line 1
$\overline{BC2}$	U3	O	Byte control line 2
$\overline{BC3}$	V2	O	Byte control line 3
$\overline{RD}$	T1	O	Read control line
RD/$\overline{WR}$	T2	O	Write control line
$\overline{ADV}$	U1	O	Address valid output
$\overline{WAIT}$/$\overline{IND}$	R5	I	Wait input / End of burst input
$\overline{BAA}$	U4	O	Burst address advance output
$\overline{CODE}$	V3	O	Code fetch status output The $\overline{CODE}$ signal has the same timing as the chip select signals.

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pin	In Out	Functions
AN[23:0]			**ADC Analog Input Port** The ADC Analog Input Port provides 24 analog input lines for the A/D converters ADC0 and ADC1.
AN0	A7	I	Analog input 0
AN1	D6	I	Analog input 1
AN2	B6	I	Analog input 2
AN3	D5	I	Analog input 3
AN4	A6	I	Analog input 4
AN5	B5	I	Analog input 5
AN6	C5	I	Analog input 6
AN7	A5	I	Analog input 7
AN8	A3	I	Analog input 8
AN9	C4	I	Analog input 9
AN10	C3	I	Analog input 10
AN11	B3	I	Analog input 11
AN12	D3	I	Analog input 12
AN13	D4	I	Analog input 13
AN14	E4	I	Analog input 14
AN15	E3	I	Analog input 15
AN16	A2	I	Analog input 16
AN17	A1	I	Analog input 17
AN18	B2	I	Analog input 18
AN19	B1	I	Analog input 19
AN20	C2	I	Analog input 20
AN21	F4	I	Analog input 21
AN22	F3	I	Analog input 22
AN23	D2	I	Analog input 23

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pin	In Out	Functions	
P0		I/O	**Port 0**[6)] Port 0 is a 16-bit bi-directional general purpose I/O port that is also used as input/output for ASC0, SSC0, CAN, GPTU, ADC0, ADC1, and the DMA Controller.	
P0.0	A10	I/O	GPT0	GPTU I/O line 0 /
		I	AD0EXTIN0	ADC0 external trigger input 0
P0.1	D7	I/O	GPT1	GPTU I/O line 1
		I	AD0EXTIN1	ADC0 external trigger input 1
		I	DMREQ0A	DMA request input 0A
P0.2	B8	I/O	GPT2	GPTU I/O line 2
		I	AD1EXTIN0	ADC1 external trigger input 0
		I	DMREQ1A	DMA request input 1A
P0.3	C7	I/O	GPT3	GPTU I/O line 3
		I	AD1EXTIN1	ADC1 external trigger input 1
P0.4	A11	I/O	GPT4	GPTU I/O line 4 /
		O	AD0EMUX0	ADC0 external multiplexer control 0
P0.5	A12	I/O	GPT5	GPTU I/O line 5
		O	AD0EMUX1	ADC0 external multiplexer control 1
P0.6	D8	I/O	GPT6	GPTU I/O line 6
		O	AD0EMUX2	ADC0 external multiplexer control 2
P0.7	B9	I/O	RXD0	ASC0 receiver input/output
P0.8	A13	O	TXD0	ASC0 transmitter output
P0.9	B10	I/O	SCLK0	SSC0 clock input/output
P0.10	A14	I/O	MRST0	SSC0 master receive input / SSC0 slave transmit output
P0.11	C9	I/O	MTSR0	SSC0 master transmit output / SSC0 slave receive input
P0.12	C10	I	RXDCAN0	CAN receiver input 0
P0.13	D10	O	TXDCAN0	CAN transmitter output 0
P0.14	A15	I	RXDCAN1	CAN receiver input 1
P0.15	B11	O	TXDCAN1	CAN transmitter output 1

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pin	In Out	Functions
P1		I/O	**Port 1**[6)] Port 1 is a 16-bit bidirectional general purpose I/O port which also serves as input or output for the GPTA.
P1.0	N18	I/O	IN0 / OUT0 line of GPTA
P1.1	R16	I/O	IN1 / OUT1 line of GPTA
P1.2	R15	I/O	IN2 / OUT2 line of GPTA
P1.3	M17	I/O	IN3 / OUT3 line of GPTA
P1.4	N16	I/O	IN4 / OUT4 line of GPTA
P1.5	P15	I/O	IN5 / OUT5 line of GPTA
P1.6	M18	I/O	IN6 / OUT6 line of GPTA
P1.7	L18	I/O	IN7 / OUT7 line of GPTA
P1.8	P16	I/O	IN8 / OUT8 line of GPTA
P1.9	U16	I/O	IN09 / OUT9 line of GPTA
P1.10	M16	I/O	IN10 / OUT10 line of GPTA
P1.11	L16	I/O	IN11 / OUT11 line of GPTA
P1.12	K17	I/O	IN12 / OUT12 line of GPTA
P1.13	K18	I/O	IN13 / OUT13 line of GPTA
P1.14	L15	I/O	IN14 / OUT14 line of GPTA
P1.15	K16	I/O	IN15 / OUT15 line of GPTA
P2		I/O	**Port 2**[6)] Port 2 is a 16-bit bidirectional general purpose I/O port which also serves as input or output for the GPTA.
P2.0	J18	I/O	IN16 / OUT16 line of GPTA
P2.1	T15	I/O	IN17 / OUT17 line of GPTA
P2.2	N15	I/O	IN18 / OUT18 line of GPTA
P2.3	J17	I/O	IN19 / OUT19 line of GPTA
P2.4	H18	I/O	IN20 / OUT20 line of GPTA
P2.5	J16	I/O	IN21 / OUT21 line of GPTA
P2.6	T16	I/O	IN22 / OUT22 line of GPTA
P2.7	M15	I/O	IN23 / OUT23 line of GPTA
P2.8	J15	I/O	IN24 / OUT24 line of GPTA
P2.9	H16	I/O	IN25 / OUT25 line of GPTA
P2.10	G18	I/O	IN26 / OUT26 line of GPTA
P2.11	H15	I/O	IN27 / OUT27 line of GPTA
P2.12	G15	I/O	IN28 / OUT28 line of GPTA
P2.13	G17	I/O	IN29 / OUT29 line of GPTA
P2.14	G16	I/O	IN30 / OUT30 line of GPTA
P2.15	F18	I/O	IN31 / OUT31 line of GPTA

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pin	In Out	Functions
P3		I/O	**Port 3**[6] Port 3 is a 16-bit bidirectional general purpose I/O port which also serves as input or output for the GPTA.
P3.0	F16	I/O	IN32 / OUT32 line of GPTA
P3.1	F15	I/O	IN33 / OUT33 line of GPTA
P3.2	E18	I/O	IN34 / OUT34 line of GPTA
P3.3	E16	I/O	IN35 / OUT35 line of GPTA
P3.4	E15	I/O	IN36 / OUT36 line of GPTA
P3.5	E17	I/O	IN37 / OUT37 line of GPTA
P3.6	D18	I/O	IN38 / OUT38 line of GPTA
P3.7	D17	I/O	IN39 / OUT39 line of GPTA
P3.8	C18	I/O	IN40 / OUT40 line of GPTA
P3.9	C17	I/O	IN41 / OUT41 line of GPTA
P3.10	B18	I/O	IN42 / OUT42 line of GPTA
P3.11	A18	I/O	IN43 / OUT43 line of GPTA
P3.12	C14	I/O	IN44 / OUT44 line of GPTA
P3.13	D15	I/O	IN45 / OUT45 line of GPTA
P3.14	B17	I/O	IN46 / OUT46 line of GPTA
P3.15	B16	I/O	IN47 / OUT47 line of GPTA
P4		I/O	**Port 4**[6] Port 4 is an 8-bit bidirectional general purpose I/O port which also serves as input/output for the GPTA or external request input for the DMA controller. During hardware reset the port 4 lines are also used as start-up configuration selection inputs and PLL clock selection inputs.
P4.0	D14	I/O	IN48 / OUT48 line of GPTA
P4.1	C13	I/O	IN49 / OUT49 line of GPTA /
		I	DMREQ0B DMA request input 0B
P4.2	A17	I/O	IN50 / OUT50 line of GPTA /]
		I	DMREQ1B DMA request input 1B
P4.3	B15	I/O	IN51 / OUT51 line of GPTA /
P4.4	C16	I/O	IN52 / OUT52 line of GPTA CFG[0]
P4.5	D13	I/O	IN53 / OUT53 line of GPTA CFG[1]
P4.6	C15	I/O	IN54 / OUT54 line of GPTA CFG[2]
P4.7	C12	I/O	IN55 / OUT55 line of GPTA / GPTA emergency shut down **CFG[2:0]: Start-up Configuration Selection Inputs** These pins are sampled during power-on reset ($\overline{\text{PORST}}$ = 0). The configuration inputs define the boot options of the TC1765 after a hardware reset operation.

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pin	In Out	Functions
P5		I/O	**Port 5[6)]** Port 5 is a 5-bit bidirectional general purpose I/O port which also serves as input or output for ASC1 and SSC1.
P5.0	D12	I/O	RXD1 ASC1 receiver input/output
		I	DMREQ0C DMA request input 0C
P5.1	B13	O	TXD1 ASC1 transmitter output
		I	DMREQ1C DMA request input 1C
P5.2	B14	I/O	SCLK1 SSC1 clock input/output
P5.3	C11	I/O	MRST1 SSC1 master receive input / SSC1 slave transmit output
P5.4	A16	I/O	MTSR1 SSC1 master transmit output / SSC1 slave receive input
TP		O	**OCDS-2 Trace Port[3)]** TP is the OCDS Level 2 Trace Port. The Trace port is only available in the TC1765T. The TP outputs are tristated during reset and deep sleep mode.
TP.0	G7	O	Trace output 0
TP.1	G8	O	Trace output 1
TP.2	H7	O	Trace output 2
TP.3	H8	O	Trace output 3
TP.4	L7	O	Trace output 4
TP.5	L8	O	Trace output 5
TP.6	M7	O	Trace output 6
TP.7	M8	O	Trace output 7
TP.8	M11	O	Trace output 8
TP.9	M12	O	Trace output 9
TP.10	L11	O	Trace output 10
TP.11	L12	O	Trace output 11
TP.12	H11	O	Trace output 12
TP.13	H12	O	Trace output 13
TP.14	G11	O	Trace output 14
TP.15	G12	O	Trace output 15
$\overline{\text{TRST}}$[7)]	R14	I	**JTAG Module Reset/Enable Input** A low level at this pin resets and disables the JTAG module. A high level enables the JTAG module.
TCK[7)]	T13	I	**JTAG Module Clock Input**
TDI[8)]	T14	I	**JTAG Module Serial Data Input**

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pin	In Out	Functions
TDO	R12	O	**JTAG Module Serial Data Output[3)]**
TMS[8)]	V15	I	**JTAG Module State Machine Control Input**
$\overline{\text{OCDSE}}$[8)]	R11	I	**OCDS Enable Input** A low level on this pin during power-on reset ($\overline{\text{PORST}}$ = 0) enables the on-chip debug support (OCDS). In addition, the level of this pin during power-on reset determines the boot configuration.
$\overline{\text{BRKIN}}$[8)]	U15	I	**OCDS Break Input** A low level on this pin causes a break in the chip's execution when the OCDS is enabled. In addition, the level of this pin during power-on reset determines the boot configuration.
$\overline{\text{BRKOUT}}$	T12	O	**OCDS Break Output[3)]** A low level on this pin indicates that a programmable OCDS event has occurred.
$\overline{\text{NMI}}$[8)]	U17	I	**Non-Maskable Interrupt Input** A high-to-low transition on this pin causes an NMI-Trap request to the CPU.
$\overline{\text{HDRST}}$[8)]	P18	I/O	**Hardware Reset Input / Reset Indication Output[6)]** Assertion of this open-drain bidirectional pin causes a synchronous reset of the chip through external circuitry. The internal reset circuitry drives this pin in response to a power-on, hardware, watchdog and power-down wake-up reset for a specific period of time. For a software reset, it is programmable whether this pin is activated or not.
$\overline{\text{PORST}}$	T17	I	**Power-on Reset Input** A low level on $\overline{\text{PORST}}$ causes an asynchronous reset of the entire chip. During power-up of the TC1765, this pin must be held active (low).
BYPASS	N17	I	**PLL Bypass Control Input** This pin is sampled during power-on reset ($\overline{\text{PORST}}$ = 0). If BYPASS is at high level, direct drive mode operation of the clock circuitry is selected and the PLL is bypassed.

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pin	In Out	Functions
XTAL1 **XTAL2**	U18 T18	I O	**Oscillator/PLL/Clock Generator Input/Output Pins** XTAL1 is the input to the oscillator amplifier and input to the internal clock generator. XTAL2 is the output of the oscillator amplifier circuit. For clocking the device from an external source, XTAL1 is driven with the clock signal while XTAL2 is left unconnected. For crystal oscillator operation XTAL1 and XTAL2 are connected to the crystal with the appropriate recommended oscillator circuitry.
ECOUT	P1	O	**EBU Clock Output**[3]
ECIN	N1		**EBU Clock Input** The ECIN pin is used to latch the data from external components into the EBU. This pin has to be connected to the ECOUT pin. Additional delay elements might be used to adapt to long delays at the address and data lines.
CPUCLK	R13	O	**CPU Clock Output**[3] General purpose clock output (can be disabled if not used). In addition, the OCDS-2 trace output data are synchronous to this clock.
$\overline{\text{TEST}}$ $\overline{\text{MODE}}$[8]	V16	I	**Test Mode Select Input** For normal operation of the TC1765 this pin should be connected to V_{DD}.
V_{DDOSC}	R18	–	**Main Oscillator Power Supply (2.5 V)**[9]
V_{SSOSC}	R17	–	**Main Oscillator Ground**
V_{DD}	J3, P2, T4, V7, U11, U13, L2, F17, D11, V9	–	**Core and EBU Power Supply (2.5 V)**[9]

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pin	In Out	Functions
V_{DDP}	L17, H17, D16, B12, C8	–	**Port 0 to 5 and Dedicated Pins Power Supply (3.3 - 5 V)[10)]**
V_{DDRAM}	G3	–	**Power Supply for PMU Memories (2.5 V)[9)]**
$V_{DDSBRAM}$	P17	–	**Power Supply for DMU Memory (2.5 V)[9)]** Used for normal and stand-by operating mode.
V_{SS}	D9, K4, K15, G9, G10, H9, H10, J7, J8, J9, J10, J11, J12, K7, K8, K9, K10, K11, K12, L9, L10, M9, M10	–	**Ground**

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pin	In Out	Functions
V_{DDM}	B4	–	**ADC Analog Part Power Supply (5 V)[10)]**
V_{SSM}	A4	–	**ADC Analog Part Ground for V_{DDM}**
V_{DDA0}	A9	–	**ADC0 Analog Part Power Supply (2.5 V)[9)]**
V_{SSA0}	B7	–	**ADC0 Analog Part Ground for V_{DDA0}**
V_{DDA1}	G4	–	**ADC1 Analog Part Power Supply (2.5 V)[9)]**
V_{SSA1}	D1	–	**ADC1 Analog Part Ground for V_{DDA1}**
V_{AREF0}	A8	–	**ADC0 Reference Voltage[10)]**
V_{AGND0}	C6	–	**ADC0 Reference Ground**
V_{AREF1}	C1	–	**ADC1 Reference Voltage[10)]**
V_{AGND1}	E2	–	**ADC1 Reference Ground[10)]**
N.C.	V17, V18	–	**Not Connected;** reserved for future expansions

1) These pins have a drive capability of 600 µA when used as outputs.

2) These pins can be connected with internal pull-up devices by setting bit SCU_CON.EBUDPEN.

3) These outputs have a drive capability of 600 µA.

4) These pins can be connected with internal pull-up devices by setting bit SCU_CON.EBUAPEN.

5) These pins can be connected with internal pull-up devices by setting bit SCU_CON.EBUCPEN.

6) These pins have a drive capability of 2.4 mA when used as outputs.

7) These pins have an internal pull-down device connected.

8) These pins have an internal pull-up device connected.

9) The voltage on power supply pins marked with 10) has to be raised earlier or at least at the same time (= time window of 1 µs) than on power supply pins marked with 9).

10) See note 9).

2 TC1765 Processor Architecture

The Central Processing Unit (CPU) of the TC1765 is based on the Infineon TriCore 32-bit microcontroller-DSP processor core architecture. It is optimized for real-time embedded systems, and combines:

- Reduced Instruction Set Computing (RISC) architecture
- Digital signal processing (DSP) operations and data structures
- Real-time responsiveness

The RISC load/store architecture provides high computational bandwidth with low system cost. Its superscalar design has three pipelines.

The TC1765 CPU is a Harvard-style architecture, with separate address and data buses for program and data memories. There are special instructions for common DSP operations and hardware-assisted data structure index generation for circular buffers (useful for filters) and bit-reversed indexing (useful for Fast Fourier Transforms). These features make it possible to efficiently analyze complex real-world signals.

The CPU's interrupt-processing architecture combines the quick responsiveness associated with microcontrollers with a high degree of interrupt-service flexibility. The architecture of the CPU minimizes interrupt latency by having few uninterruptable multi-cycle instructions, by supporting fast context switching, and supporting task-based memory protection.

While the TriCore architecture employs 32-bit Instruction formats, frequently-used instructions have an optional 16-bit instruction format. This results in smaller code size, and faster code bandwidth. Additional benefits of this approach include lowered program memory requirements, lower system cost, and less power consumption.

2.1 Central Processing Unit

This section provides an overview of the TC1765 Central Processing Unit (CPU) architecture. The basic features include the following.

- Data paths: 32 bits throughout
- Address space: 4 Gigabytes, unified, for data, program, and I/O
- Instruction formats: mixed 32-bit and 16-bit formats
- Low interrupt latency and flexible interrupt prioritization scheme
- Fast automatic context switching
- Separate multiply-accumulate unit
- Saturating integer arithmetic
- Bit-handling operations
- Packed-data operations
- Zero-overhead looping
- Flexible power management
- Byte and bit addressing
- Little-endian byte ordering
- Precise exceptions

Figure 2-1 illustrates the architecture of the TC1765's Central Processing Unit (CPU). It is comprised of an Instruction Fetch Unit, an Execution Unit, a General Purpose Register File, and several peripheral interfaces.

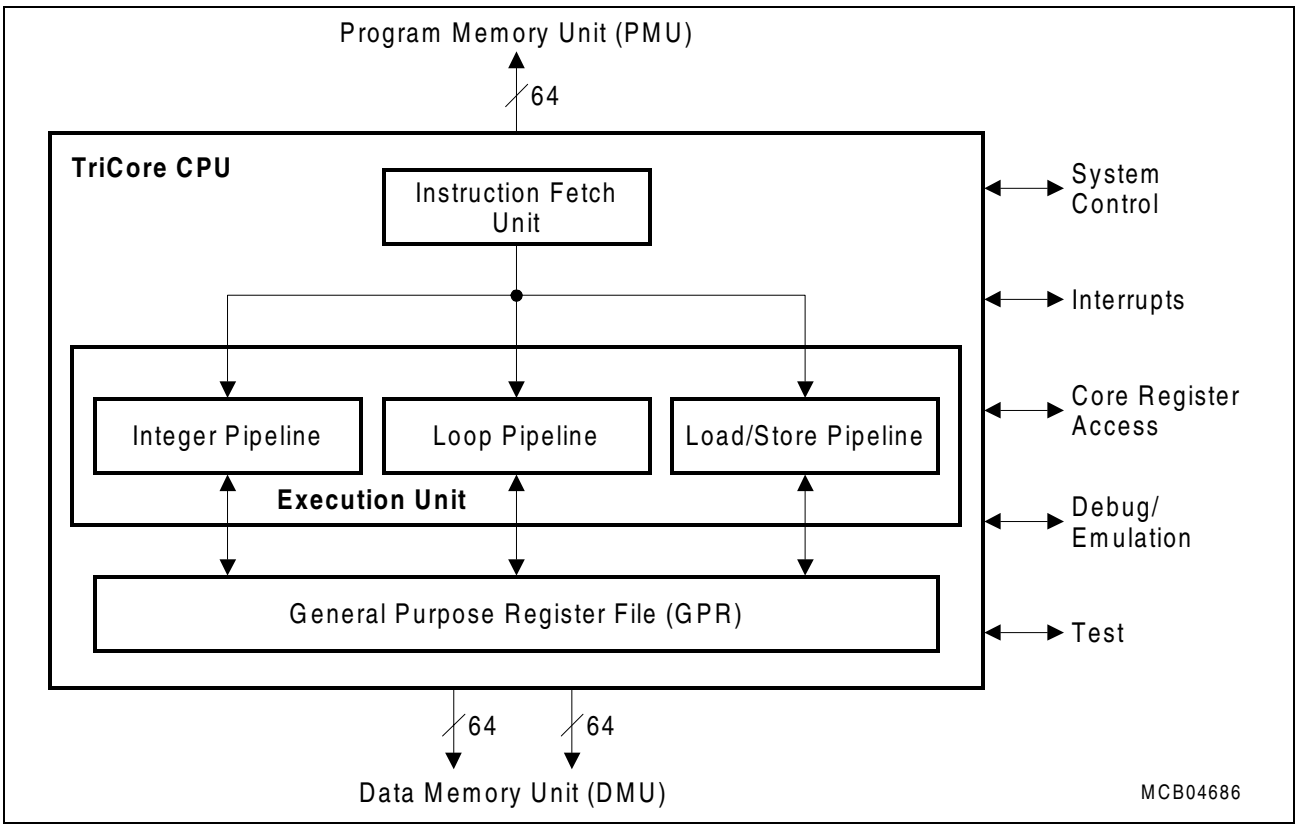


Figure 2-1 Central Processing Unit (CPU) Block Diagram

2.1.1 Instruction Fetch Unit

Figure 2-2 shows the Instruction Fetch Unit. It prefetches and aligns incoming instructions from the 64-bit wide Program Memory Unit (PMU). The Issue Unit directs the instruction to the appropriate pipeline. The Instruction Protection Unit checks the validity of accesses to the PMU and also checks for instruction breakpoint conditions. The PC Unit is responsible for updating the issue and prefetch program counters.

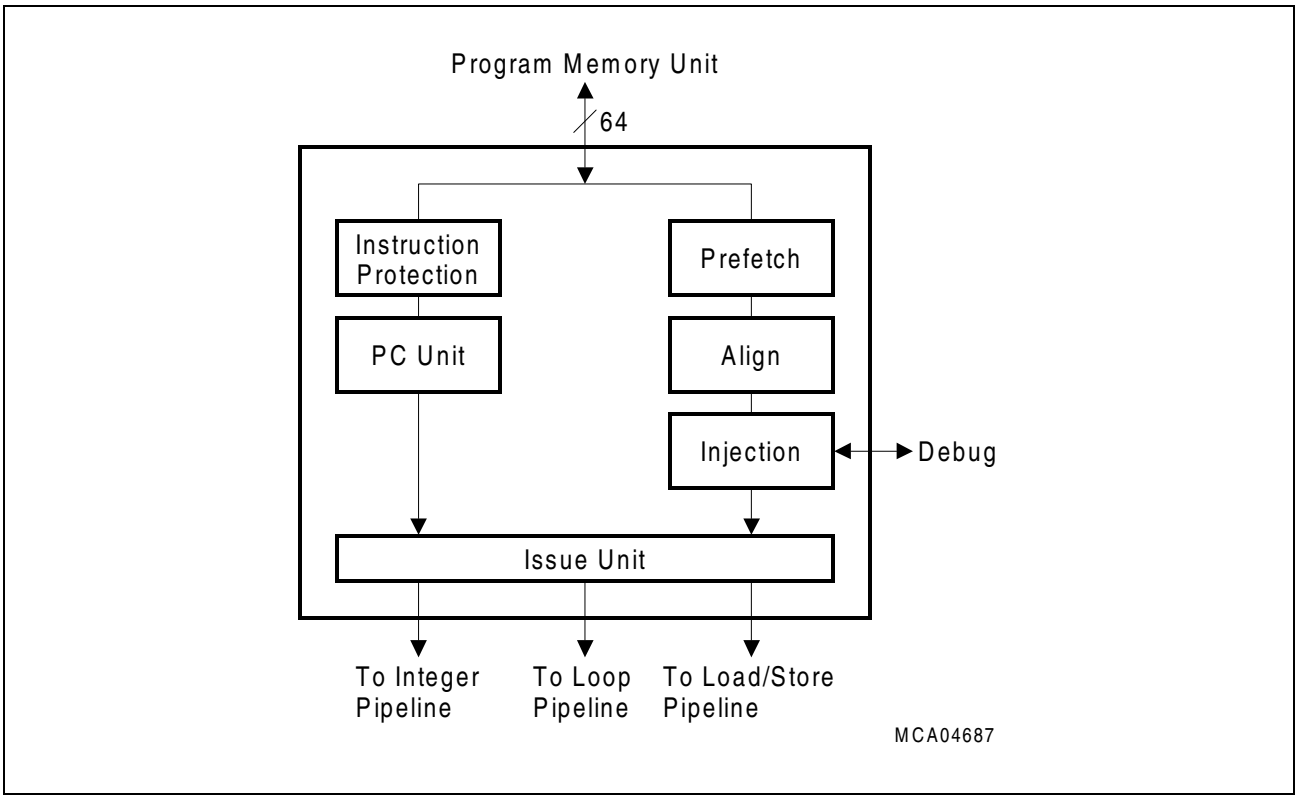


Figure 2-2 Instruction Fetch Unit

2.1.2 Execution Unit

As shown in **Figure 2-3**, the Execution Unit contains the Integer Pipeline, the Loop Pipeline, and the Load/Store Pipeline.

The Integer Pipeline and Load/Store Pipeline have four stages: Fetch, Decode, Execute, and Write-back. The Execute stage may extend beyond one cycle to accommodate multi-cycle operations such as load instructions.

The Loop Pipeline has two stages: Decode and Write-back.

All three pipelines operate in parallel, permitting up to three instructions to execute in one clock cycle.

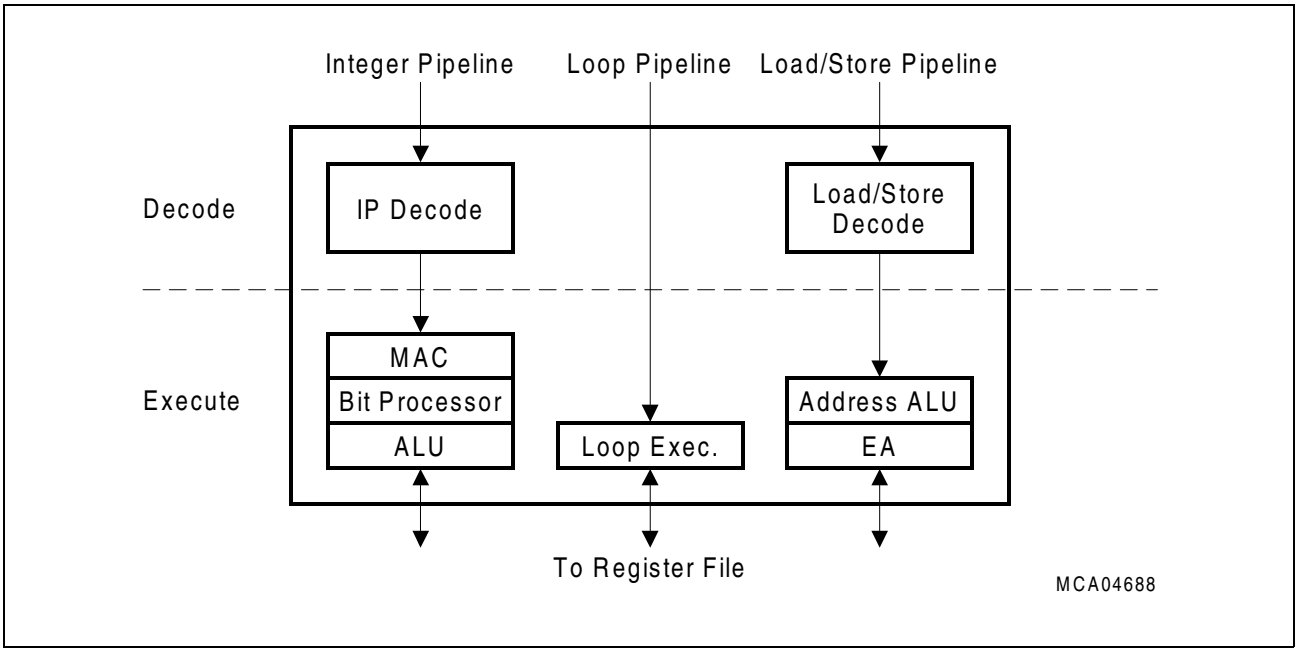


Figure 2-3 Execution Unit

Figure 2-3 introduces the following acronyms and abbreviations:

- IP Decode - Instruction Prefetch and Decode
- MAC - Multiply-Accumulate Unit
- ALU - Arithmetic/Logic Unit
- Loop Exec. - Loop Execution Unit
- EA - Effective Address

2.1.3 General Purpose Register File

The CPU has a General Purpose Register (GPR) file, divided into an Address Register File (registers A0 through A15) and a Data Register File (registers D0 through D15).

The data flow for instructions issued to/from the Load/Store Pipeline is steered through the Address Register File. The data flow for instructions issued to/from the Integer Pipeline and for data load/store instructions issued to/from the Load/Store Pipeline is steered through the Data Register File.

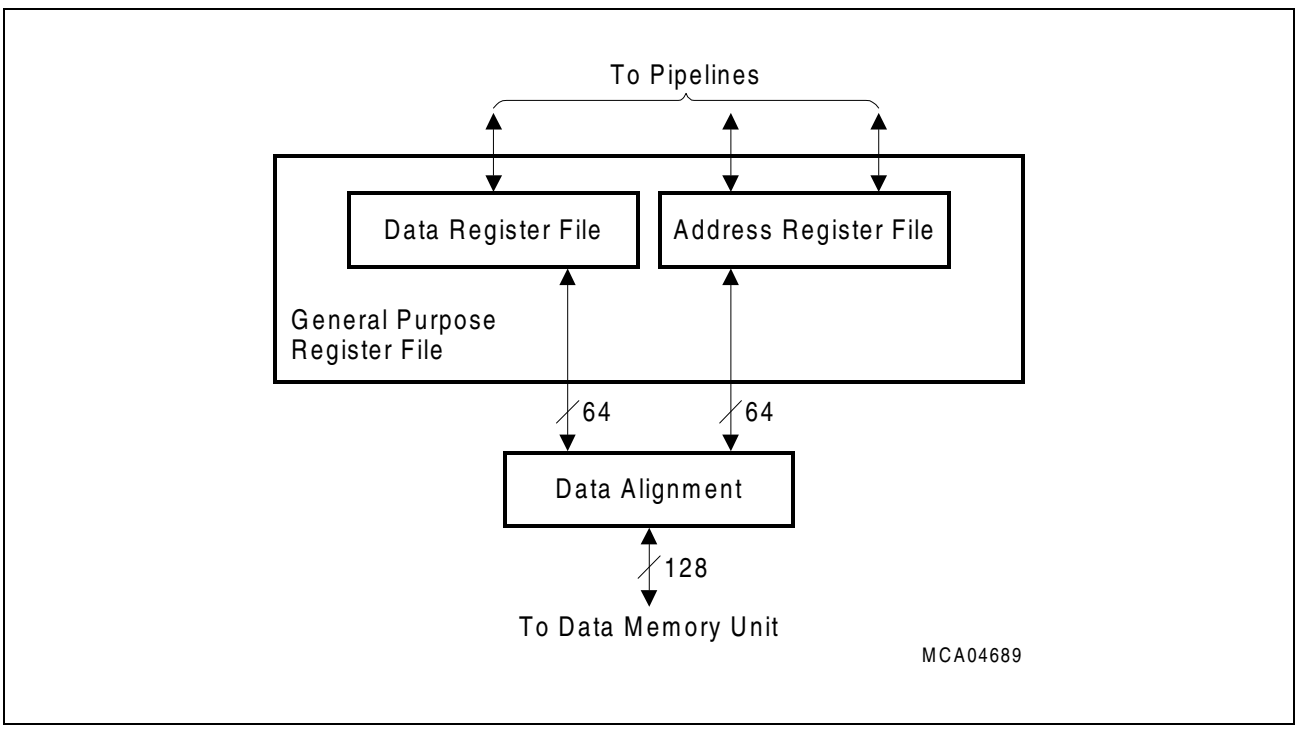


Figure 2-4 General Purpose Register File

2.1.4 Program State Registers

The program state registers consist of 32 General Purpose Registers (GPRs), two 32-bit registers with program status information (PCXI and PSW), and a Program Counter (PC). PCXI, PSW, and PC are Core Special Function Registers (CSFRs).

As shown in **Figure 2-5**, the 32 General Purpose Registers are divided into sixteen 32-bit data registers (D0 through D15) and sixteen 32-bit address registers (A0 through A15).

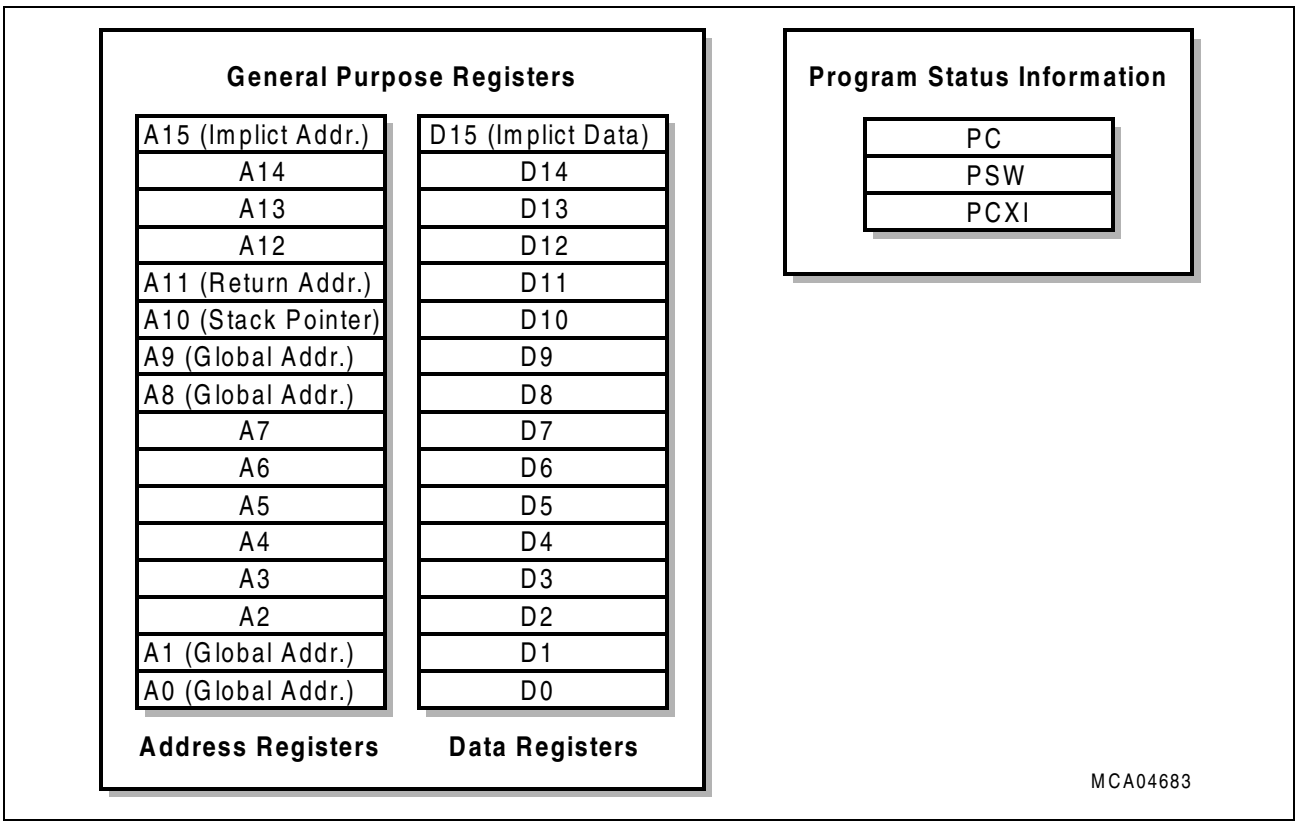


Figure 2-5 Program State Registers

Four GPRs have special functions: D15 is used as an implicit data register, A10 is the Stack Pointer (SP), A11 is the return address register, and A15 is the implicit address register.

Registers 0-7 are called the lower registers and 8-15 are called the upper registers.

Registers A0 and A1 in the lower address registers and A8 and A9 in the upper address registers are defined as system global registers. These registers are not included in either context partition, and are not saved and restored across calls or interrupts.The operating system normally uses them to reduce system overhead.

The PCXI and PSW registers contain status flags, previous execution information, and protection information.

2.1.5 Data Types

The TriCore instruction set supports operations on booleans, bit-strings, characters, signed fractions, addresses, signed and unsigned integers, and single-precision floating-point numbers. Most instructions work on a specific data type, while others are useful for manipulating several data types.

2.1.6 Addressing Modes

Addressing modes allow load and store instructions to efficiently access simple variables and data elements within data structures such as records, randomly and sequentially accessed arrays, stacks, and circular buffers. Simple variables and data elements are 1, 8, 16, 32 or 64 bits wide.

Addressing modes provide efficient compilation of programs written in the C programming language, easy access to peripheral registers, and efficient implementation of typical DSP data structures. Hardware-assisted DSP data structures include circular buffers for filters and bit-reversed indexing for FFTs. The following seven addressing modes are supported in the TriCore architecture:

- Absolute
- Base + Short Offset
- Base + Long Offset
- Pre-increment or pre-decrement
- Post-increment or post-decrement
- Circular (modulo)
- Bit-Reverse

2.1.7 Instruction Formats

The CPU architecture supports both 16-bit and 32-bit instruction formats. All instructions have a 32-bit format. The 16-bit instructions are a subset of the 32-bit instructions, chosen because of their frequency of use and included to reduce code space.

2.1.8 Tasks and Contexts

Throughout this document, the term *task* refers to an independent thread of control: Software-managed Tasks (SMTs) and Interrupt Service Routines (ISRs).

Software-managed tasks are created through the services of a real-time kernel or operating system and dispatched under the control of scheduling software. Interrupt Service Routines (ISRs) are dispatched by hardware in response to an interrupt. An ISR is the code that is invoked by the processor directly on receipt of an interrupt. Software-managed tasks are sometimes referred to as user tasks, assuming that they will execute in User Mode.

Each task is allocated its own permission level. The individual permissions are enabled or disabled primarily by I/O mode bits in the Program Status Word (PSW).

The processor state associated with a task is called the task's *context*. The context includes everything the processor needs in order to define the current state of the task. The system saves the current task's context when another task is about to run, and restores the task's context when the task is to be resumed. The context includes the Program State Registers. The CPU efficiently manages and maintains the contexts of the tasks through hardware.

2.1.8.1 Upper and Lower Contexts

The context is subdivided into the Upper Context and the Lower Context, as illustrated in **Figure 2-6**. The Upper Context consists of the upper address registers, A10 – A15, and the upper data registers, D8 – D15. These registers are designated as non-volatile, for purposes of function calling. The Upper Context also includes the PCXI and PSW registers. The Lower Context consists of the lower address registers, A2 through A7, the lower data registers, D0 through D7, and saved PC, and again the PCXI register.

Both Upper and Lower Contexts include a Link Word contained in register PCXI. Contexts are saved in fixed-size memory areas (see **Section 2.1.8.2**); they are linked together *via* the link word.

The Upper Context is saved automatically on Interrupts. It is also saved on CALL instructions and restored on RETURN instructions. The Lower Context must be saved and restored by the IISR if the ISR needs to use more registers than are available in the Upper Context.

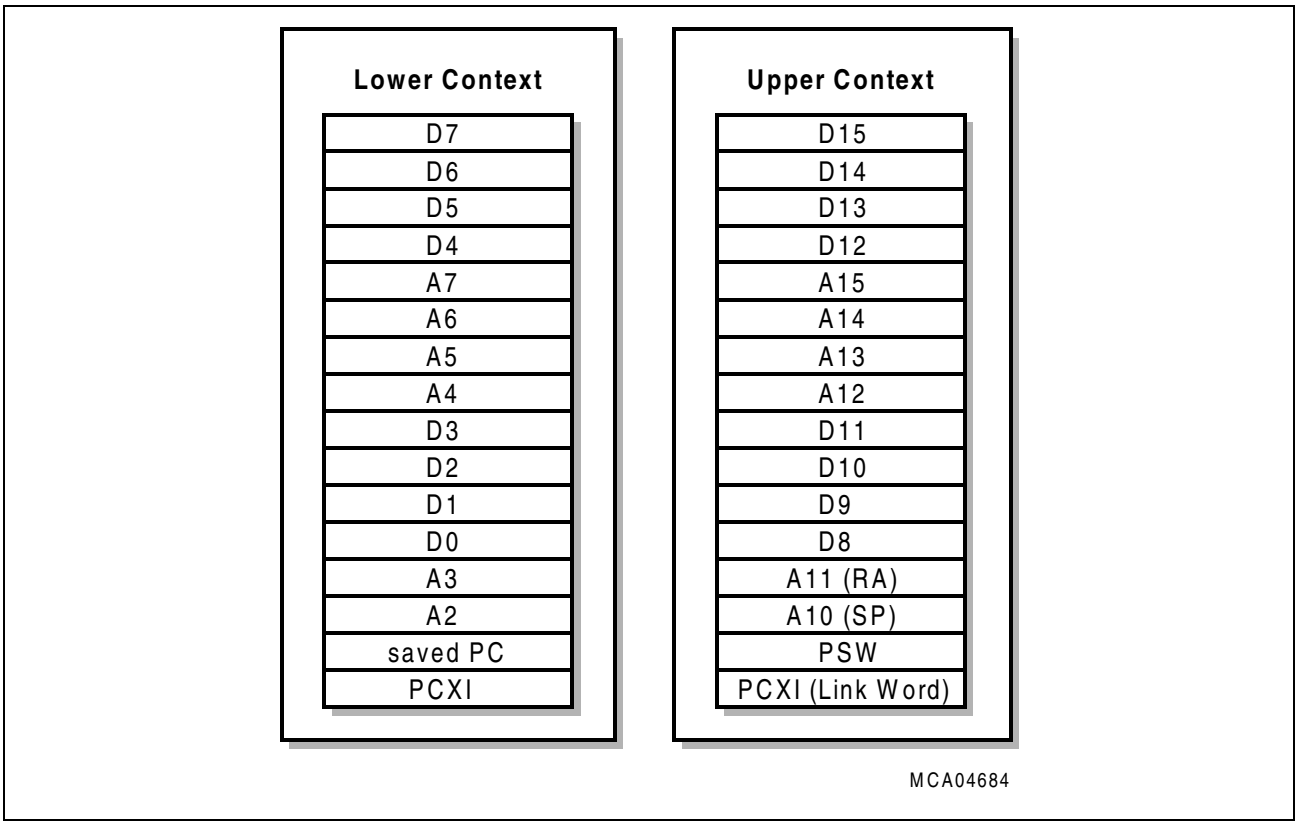


Figure 2-6 Upper and Lower Contexts

2.1.8.2 Context Save Areas

The architecture uses linked lists of fixed-size Context Save Areas (CSAs) which accommodate systems with multiple interacting threads of control. A CSA is sixteen words of memory storage, aligned on a 16-word boundary. A single CSA can hold exactly one Upper or one Lower Context. Unused CSAs are linked together on a free list. They are allocated from the free list as needed and returned to it when no longer needed. Allocation and freeing are handled transparently by the processor. They are transparent to the applications code. Only system initialization code and certain operating system exception-handling routines need to access the CSAs or their lists explicitly. The number of CSAs that can be used is limited only by the size of the available data memory.

Note: In the TC1765, Context Save Areas can only be located either in the local data scratch-pad RAM (SPRAM) or in external memory in the cacheable Segment 10.

2.1.8.3 Fast Context Switching

The TC1765 CPU uses a uniform context-switching method for function calls, interrupts, and traps. In all cases, Upper Context of the task is automatically saved and restored by hardware. Saving and restoring of the Lower Context is left as an option for the new task. An explanation of CPU management of the contexts can be found in **Section 2.2.2**.

Fast context switching is further enhanced by the TriCore's unique memory subsystem

design, which allows a complete Upper or Lower Context to be saved in as little as two clock cycles.

2.1.9 Interrupt System

An interrupt request can be generated by the TC1765 on-chip peripheral units or it can be generated by external events. Requests can be targeted to the CPU.

In order to better differentiate the programmable stages of interrupt processing available in the TC1765, this document refers to an interrupt-triggering event as an Interrupt Service Request. The TC1765 interrupt system evaluates service requests and the highest-priority service request is then presented to the CPU by way of an interrupt.

For a CPU interrupt, the entry code for the Interrupt Service Routine (ISR) is contained in an Interrupt Vector Table. Each entry in this table corresponds to a fixed-size code block. (If an ISR requires more code than fits in an entry, it must include a jump instruction to vector it to the rest of the ISR elsewhere in memory.) Each interrupt source is assigned an interrupt priority number. All priority numbers are programmable. The ISR uses the priority number to determine the location of the entry code block.

The prioritization of service routines enables nested interrupts and the use of interrupt priority groups. See **Chapter 14** for more information.

2.1.10 Trap System

Trap events break the normal execution of code much like interrupts. But traps are different from interrupts in these ways:

- Trap Service Routines (TSR) reside in the Trap Vector Table, separate from the Interrupt Vector Table.
- A trap does not change the CPU's interrupt priority.
- Traps cannot be disabled by software, and are always active.

A trap occurs as a result of an exception within one of the following classes of events.

- Reset
- Internal protection
- Instruction errors
- Context management
- Internal bus and peripheral errors
- Assertion
- System call
- Non-maskable interrupt

Each entry in the Trap Vector Table corresponds to a fixed-size code block. (If a TSR requires more code than fits in an entry, it must include a jump instruction to vector it to the rest of the TSR located elsewhere in memory.) When a trap is taken, its Trap Identification Number (TIN) is placed in data register D15. The trap handler uses the TIN to identify the cause of the trap. During trap arbitration, the pending trap with the lowest TIN will be chosen to execute. See **Chapter 15** for more information.

2.1.11 Protection System

There are two protection systems in the TC1765. A memory-access protection system protects code and data memory regions, as described in **Section 2.1.11.1** and **Section 2.1.11.2**. Access to sensitive system registers is protected by hardware against system malfunctions, as described in **Section 2.1.11.3**.

2.1.11.1 Permission Levels

Each task can be assigned a specific permission level. Individual permissions are enabled through the I/O Mode bits in the Program Status Word (PSW). The three permission levels are listed here, in decreasing order of restrictiveness.

- **User-0 Mode**
  - Used for tasks that do not access peripheral devices.
  - Tasks at this level do not have permission to enable or disable interrupts.
- **User-1 Mode**
  - Used for tasks that access common, unprotected peripherals.
  - Accesses typically include read/write accesses to serial ports and read accesses to timers and most I/O status registers.
  - Tasks at this level may disable interrupts.
- **Supervisor Mode**
  - Permits read/write access to system registers and all peripheral devices.
  - Tasks at this level may disable interrupts.

2.1.11.2 Memory Protection Model

The Memory Protection Model of the CPU is based on address ranges, where each address range has an associated permission setting. Address ranges and their associated permissions are specified in identical sets of tables residing in the Core Special Function Register (CSFR) space. Each set is referred to as a Protection Register Set (PRS).

The TC1765 incorporates two sets of Protection Register Sets each for code and data memory. The number of sets is implementation-specific. Other TriCore products may have implemented a different number (up to four) of Protection Register Sets.

When the protection system is enabled the CPU checks every load/store or instruction fetch address before performing the access. Legal addresses must fall within one of the

ranges specified in the currently selected PRS, and permission for that type of access must be present in the matching range.

2.1.11.3 Watchdog Timer and ENDINIT Protection

Registers that control basic TC1765 configuration and operation can be protected via a special End-of-Initialization (ENDINIT) bit. The ENDINIT bit globally protects those TC1765 registers that control basic system configuration against unintentional modification. Write accesses to registers protected via this ENDINIT-bit are prohibited as long as this bit is set to 1. To clear the bit and to enable access to these registers again, a special password-protected access sequence to the Watchdog Timer registers must be performed. The bit must be set to 1 again within a defined time-out period, otherwise a system malfunction is assumed to have occurred, and the Watchdog Timer triggers a reset of the TC1765. See **Chapter 18** for more details.

2.1.12 Reset System

Several events will cause the TC1765 system to be reset:

- **Power-On Reset**
  - Activated through an external pin when the power to the device is turned on (also called cold reset)
- **Hard Reset**
  - Activated through an external pin ($\overline{\text{HDRST}}$) during run time (also called warm reset)
- **Soft Reset**
  - Activated through a software write to a reset-request register, which has a special protection mechanism to prevent accidental access
- **Watchdog Timer Reset**
  - Activated through an error condition detected by the Watchdog Timer
- **Wake-up Reset**
  - Activated through an external pin to wake the device from a power saving mode

A status register allows the CPU to check which of the triggers caused the reset.

2.2 Processor Registers

The processor contains general purpose registers to store instruction operands. It has special purpose registers for managing the state of the processor itself.

The CPU's operations are controlled by a set of Core Special Function Registers (CSFRs). These registers also provide status information about its operation. The CSFRs are split into the following groups:

- Program State Information
- Context Management
- Stack Management
- Interrupt and Trap Control
- System Control
- Memory Protection
- Debug Control

The following sections summarize these registers. The CSFRs are complemented by a set of General Purpose Registers (GPRs). **Table 2-1** shows all CSFRs and GPRs.

Table 2-1 Core Register Map

Register Name	Description
D0 – D15	General Purpose Data Registers
A0 – A15	General Purpose Address Registers
PSW	Program Status Word
PCXI	Previous Context Information Register
PC	Program Counter
FCX	Free CSA List Head Pointer
LCX	Free CSA List Limit Pointer
ISP	Interrupt Stack Pointer
ICR	ICU Interrupt Control Register
BIV	Interrupt Vector Table Pointer
BTV	Trap Vector Table Pointer
SYSCON	System Configuration Register
DPRx_0 – DPRx_3	Data Segment Protection Registers for Set x (x = 0, 1)
CPRx_0 – CPRx_1	Code Segment Protection Registers for Set x (x = 0, 1)
DPMx_0 – DPMx_3	Data Protection Mode Register for Set x (x = 0, 1)
CPMx_0 – CPMx_1	Code Protection Mode Register for Set x (x = 0, 1)
DBGSR	Debug Status Register

Table 2-1 Core Register Map (cont'd)

Register Name	Description
EXEVT	External Break Input Event Specifier
SWEVT	Software Break Event Specifier
CREVT	Core SFR Access Event Specifier
TRnEVT	Trigger Event n Specifier (n = 0, 1)

The CPU accesses the CSFRs through two instructions: MFCR and MTCR. The MFCR instruction (Move From Core Register) moves the contents of the addressed CSFR into a data register. MFCR can be executed on any privilege level. The MTCR instruction (Move To Core Register) moves the contents of a data register to the addressed CSFR. To prevent unauthorized writes to the CSFRs, the MTCR instruction can be executed on Supervisor privilege level only.

The CSFRs are also mapped into the top of Segment 15 in the memory address space. This mapping makes the complete architectural state of the CPU visible in the address map. This feature provides efficient debug and emulator support.

*Note: The CPU is **not allowed** to access the CSFRs through this mechanism — it must use the MFCR and MTCR instructions. Trying to access the CSFRs through normal load and store instructions results in a MEM trap.*

The instruction set provides no single-bit, bit field, or load-modify-store accesses to the CSFRs. The only other instruction affecting a CSFR, is the RSTV instruction (Reset Overflow Flags), which resets only the overflow flags in the PSW, without modifying any of the other PSW bits. This instruction can be executed at any privilege level.

Note: Access to the Core SFRs through their mapped addresses in segment 15 is implemented primarily for debug purposes. Special attention needs to be paid when accessing these registers. It is strongly advised to not write to the CSFRs while the core is executing. Reading the registers while the core is running does not guarantee coherent status information.
A mid-range or high-range emulator can use the external bus as a fast route to the internal FPI Bus. However, certain restrictions are placed on this mode of operation regarding access to the CSFRs and GPRs: The external bus cannot be used to access state in the core (GPRs and CSFRs) while the core is running (not halted) and is configured to perform accesses to the external bus.

Figure 2-7 shows the General Purpose Registers (GPRs). The 32-bit wide GPRs are split evenly into sixteen data registers, or DGPRs, (D0 to D15) and sixteen address registers, or AGPRs, (A0 to A15). Separation of data and address registers facilitates efficient performance of arithmetic and memory operations in parallel. Several instructions interchange information between data and address registers in order, for example, to create or derive table indexes. 64-bit values can be represented by

concatenating two consecutive double-word-aligned data registers. Eight such extended-size registers (E0, E2, E4, E6, E8, E10, E12, and E14) are available.

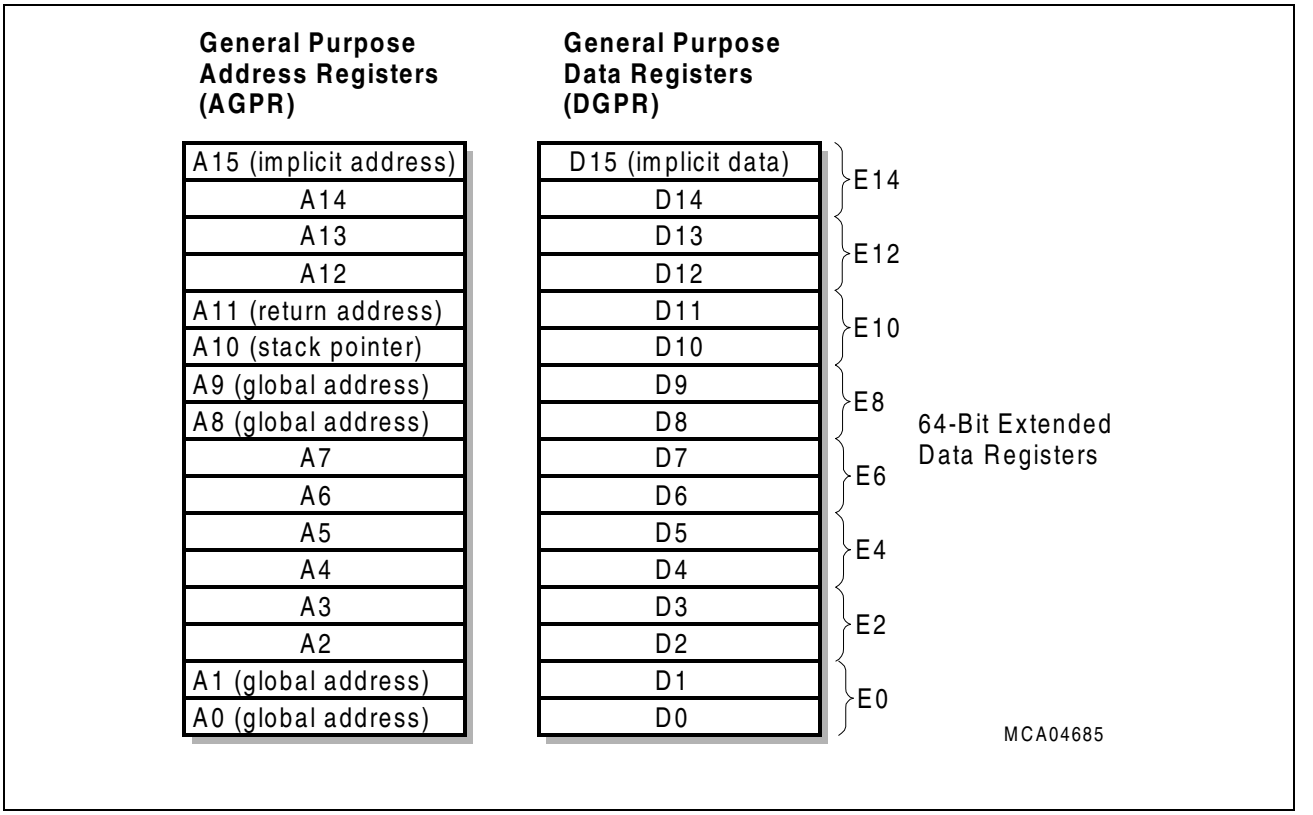


Figure 2-7 General Purpose Registers (GPRs)

As shown in **Figure 2-7**, registers A0, A1, A8, and A9 are defined as System Global Registers. Their contents are not saved and restored across calls, traps, or interrupts. Register A10 is used as the Stack Pointer (SP) register. A11 is used to store the return address (RA) for calls and linked jumps and to store the return program counter (PC) value for interrupts and traps as part of the Upper Context.

The 32-bit instructions have unlimited use of the GPRs. However, many 16-bit instructions implicitly use A15 as their address register and D15 as their data register to make the encoding of these instructions into 16 bits possible.

There are no separate floating-point registers — the data registers are used to perform floating-point operations. Floating-point data is saved and restored automatically using the fast context-switching capabilities of the TC1765.

The GPRs are an essential part of a task's context. When saving or restoring a task's context to and from memory, the context is split into the Upper Context and Lower Context as shown in **Figure 2-6**. Registers A2 through A7 and D0 through D7 are part of the Lower Context. Registers A10 through A15 and D8 through D15 are part of the Upper Context.

2.2.1 Program State Information Registers

The PC, PSW, and PCXI registers hold and reflect Program State Information. When saving and restoring a task's context, the contents of these registers are saved and restored or modified during this process.

2.2.1.1 Program Counter (PC)

The Program Counter (PC) holds the address of the instruction which is currently fetched and forwarded to the CPU pipelines. The CPU handles updates of the PC automatically.

Software can use the current value of the PC for various tasks, such as performing code address calculations. Reading the PC through software executed by the CPU must only be done with an MFCR instruction. Explicit writes to the PC through an MTCR instruction must not be done due to possible unexpected behavior of the CPU.

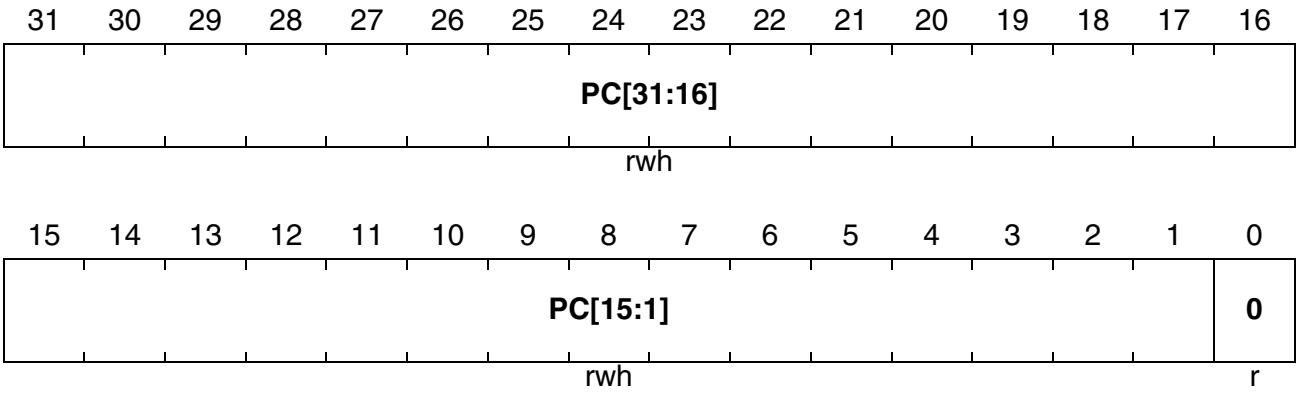
Note: The CPU must not perform Load/Store instructions to the mapped address of the PC in Segment 15. A MEM trap will be generated in such a case.

Note: Reading the PC while the Core is executing, either through an MFCR instruction or via its mapped address in Segment 15 (see below), will return a value which is representative of where the code is currently executed from, however, it is not guaranteed that the value returned will always correspond to an instruction that has been or will be executed. For example, it is possible for the PC to point to the target of a predicted branch which is subsequently resolved as mispredicted. Thus, the branch target instruction will not be executed; however, it should be possible to implement a statistical profile/coverage report with some degree of error by sampling the PC value while the CPU is running.

In Debug Mode, explicit read and write operations to the PC can be performed using its mapped address in Segment 15. This must only be done through an FPI Bus master other than the CPU itself (through the DMU). Several restrictions apply to this operation:

- *Writing to the PC while the Core is executing* is non-deterministic and the user is strongly advised not to do so. The correct sequence the user should adopt is: halt the Core, modify the PC, remove Core from Halt mode.
- *Reading the PC while the Core is halted* will return the PC of the first instruction to be executed once the Core is released from Halt mode. The only exception to this is if an interrupt or asynchronous trap is received by the Core immediately after it is removed from Halt mode prior to the first instruction being executed.
- *Writing to the PC while the Core is halted* will modify the PC in a deterministic way. the new value will be the PC of the first instruction to be executed once the Core is released from Halt mode. The only exception to this is if an interrupt or asynchronous trap is received by the Core immediately after it is removed from Halt mode prior to the first instruction being executed.

PC
Program Counter **Reset Values: Boot ROM Boot: BFFF FFFC$_H$**
External Memory Boot: A000 0000$_H$
Emulator Boot: BE00 0000$_H$



Field	Bits	Type	Description
PC	[31:1]	rwh	**Program Counter**
0	0	r	**Reserved**

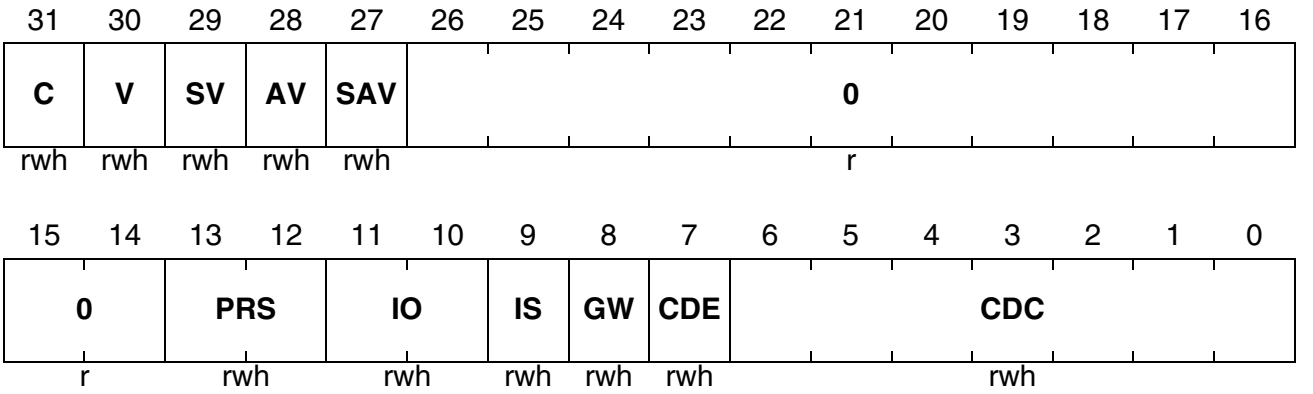
Note: Bit 0 of the PC register is a read-only bit, hard-wired to 0. This ensures that only half-word aligned addresses can be placed into the PC (instructions can only be aligned to half-word addresses).

2.2.1.2 Program Status Word (PSW)

The Program Status Word (PSW) register holds the instruction flags and the control bits for a number of options of the overall protection system.

A special instruction is available that affects only the overflow flag bits in register PSW. The RSTV (Reset Overflow Flags) instruction clears bits V, SV, AV and SAV in PSW without modifying any other PSW bit.

PSW
Program Status Word **Reset Value: 0000 0B80$_H$**


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
C V SV AV SAV 0
rwh rwh rwh rwh rwh r
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 PRS IO IS GW CDE CDC
r rwh rwh rwh rwh rwh rwh

Field	Bits	Type	Description
CDC	[6:0]	rwh	**Call Depth Counter Field** The CDC field consists of two variable-width fields. The first is a mask field, consisting of a string of zero or more initial 1 bits, terminated by the first 0 bit. The remaining bits of the field are the call depth counter. 0ccccc$_B$ 6-bit counter; trap on overflow 10cccc$_B$ 5-bit counter; trap on overflow 110ccc$_B$ 4-bit counter; trap on overflow 1110cc$_B$ 3-bit counter; trap on overflow 11110c$_B$ 2-bit counter; trap on overflow 111110$_B$ 1-bit counter; trap on overflow 111111$_B$ Trap every call (call trace mode) 1111111$_B$ Disable call depth counting When the call depth counter overflows, a trap is generated. Depending on the width of the mask field, the call depth counter can be set to overflow at any power of two boundary, from 1 to 64. Setting the mask field to 1111110$_B$ allows no bits for the counter, and causes every call to be trapped. This is used for call tracing. Setting the field to mask field to 1111111$_B$ disables call depth counting altogether.

Field	Bits	Type	Description
CDE	7	rwh	**Call Depth Count Enable** The CDE bit enables call-depth counting, provided that the CDC mask field is not all 1's. CDE is set to 1 by default, but should be cleared by the SYSCALL instruction Trap Service Routine to allow a trapped SYSCALL instruction to execute without producing another trap upon return from the trap handler. It is then set again when the next SYSCALL instruction is executed. 0 Call depth counter disabled 1 Call depth counter enabled
GW	8	rwh	**Global Register Write Permission** GW controls whether the current execution thread has permission to modify the global address registers. Most tasks and ISRs will use the global address registers as "read only" registers, pointing to the global literal pool and key data structures. However, a task or ISR can be designated as the "owner" of a particular global address register, and is allowed to modify it. The system designer must determine which global address variables are used with sufficient frequency and/or in sufficiently time-critical code to justify allocation to a global address register. By compiler convention, global address register A0 is reserved as the base register for short form loads and stores. Register A1 is also reserved for compiler use. Registers A8 and A9 are not used by the compiler, and are available for holding critical system address variables. 0 Write permission to global registers A0, A1, A8, and A9 is disabled 1 Write permission to global registers A0, A1, A8, and A9 is enabled

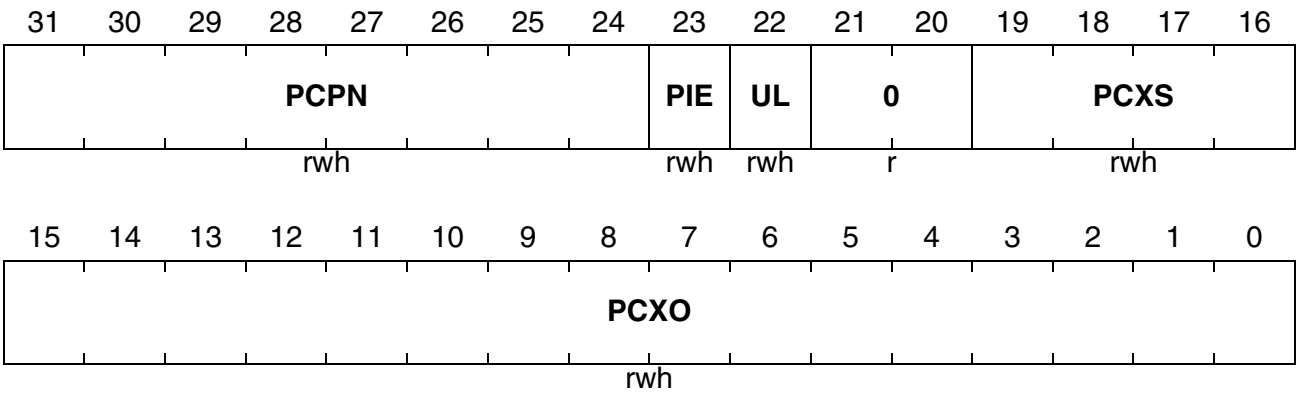
Field	Bits	Type	Description
IS	9	rwh	**Interrupt Stack Control** Determines whether the current execution thread is using the shared global (interrupt) stack or a user stack. 0 U**ser Stack**. If an interrupt is taken when the IS bit is 0, then the stack pointer register is loaded from the ISP register before execution starts at the first instruction of the Interrupt Service Routine. 1 S**hared Global Stack**. If an interrupt is taken when the IS bit is 1, then the current value of the stack pointer register is used by the Interrupt Service Routine.
IO	[11:10]	rwh	**Access Privilege Level Control** This 2-bit field selects determines the access level to special function registers and peripheral devices. 00_B **User-0 Mode**: No peripheral access. Access to segments 14 and 15 is prohibited and will result in a trap. This access level is given to tasks that need not directly access peripheral devices. Tasks at this level do not have permission to enable or disable interrupts. 01_B **User-1 Mode**: regular peripheral access. This access level enables access to common peripheral devices that are not specially protected, including read/write access to serial I/O ports, read access to timers, and access to most I/O status registers. Tasks at this level may disable interrupts. 10_B **Supervisor Mode**. This access level enables access to all peripheral devices. It enables read/write access to core registers and protected peripheral devices. Tasks at this level may disable interrupts. 11_B **Reserved**; this encoding is reserved and is not defined.

Field	Bits	Type	Description
PRS	[13:12]	rwh	**Protection Register Set Selection** The PRS field selects one of two possible sets of memory protection register values controlling load and store operations and instruction fetches within the current process. This field indicates the current protection register set. 00 Protection register set 0 selected 01 Protection register set 1 selected 10 Reserved; don't use this combination 11 Reserved; don't use this combination
0	[26:14]	r	**Reserved**; read as 0; should be written with 0;
SAV	27	rwh	**Sticky Advance Overflow Flag** This flag is set whenever the advanced overflow flag is set. It remains set until it is explicitly cleared by an RSTV (Reset Overflow bits) instruction.
AV	28	rwh	**Advance Overflow Flag** This flag is updated by all instructions that update the overflow flag and no others. This flag is determined as the boolean exclusive of the two most significant bits of the result.
SV	29	rwh	**Sticky Overflow Flag** This flag is set when an overflow occurs. This flag remains set until it is explicitly reset by an RSTV (Reset Overflow bits) instruction.
V	30	rwh	**Overflow Flag** This flag is set when an overflow occurs.
C	31	rwh	**Carry Flag** This flag is set when a carry occurs.

2.2.1.3 Previous Context Information Register (PCXI)

This register holds information about the previous task's context, and is saved and restored together with both the Upper and the Lower Context. It also contains the Previous Context Pointer (PCX), which holds the address of the previous task's context save area (CSA).

PCXI
Previous Context Information Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PCXO	[15:0]	rwh	**Previous Context Pointer Offset Field** The combined PCXO and PCXS fields form the pointer PCX, which points to the CSA of the previous context.
PCXS	[19:16]	rwh	**PCX Segment Address** This field contains the segment address portion of the PCX.
0	20, 21	r	**Reserved**; read as 0; should be written with 0;
UL	22	rwh	**Upper/Lower Context Tag** The UL context tag bit identifies the type of context saved. 0 Lower Context 1 Upper Context If the type does not match the type expected when a context restore operation is performed, a trap is generated.
PIE	23	rwh	**Previous Interrupt Enable** PIE indicates the state of the interrupt enable bit (ICR.IE) for the interrupted task.
PCPN	[31:24]	rwh	**Previous CPU Priority Number** This bit field contains the priority level number of the interrupted task.

2.2.2 Context Management Registers

The Context Management Registers (CMR) are comprised of three pointer registers, FCX, PCX, and LCX. These pointers handle context management and are used during context save/restore operations.

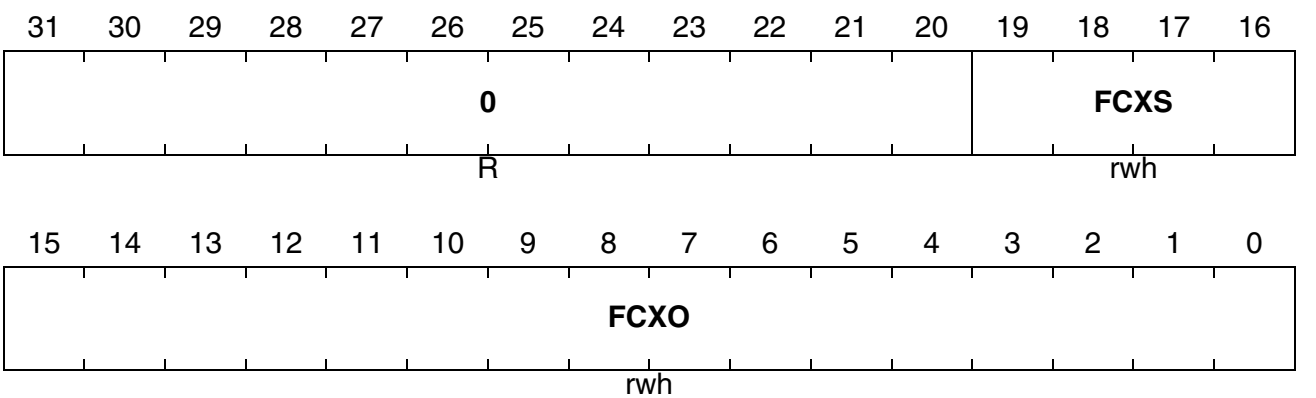
Each pointer register consists of two fields: a 16-bit offset and a 4-bit segment specifier. A Context Save Area (CSA) is an address range containing sixteen word locations (64 bytes). Each CSA can save one Upper Context or one Lower Context. Incrementing a CMR pointer offset value by 1 will point it at the CSA that is sixteen word locations above the previous one.

The FCX pointer register points to the head of the CSA free list. The previous context pointer (PCX) points to the CSA of the previous task. PCX is part of the previous context information register PCXI. The LCX pointer register is used to recognize impending CSA list underflows. If the value of FCX used on an interrupt or CALL instruction matches the limit value, the context-save operation will be completed, but the target address will be forced to the trap vector address that handles CSA list depletion.

2.2.2.1 Free Context List Head Pointer (FCX)

The FCX register points to the address of the next available context save area (CSA) in the linked list of CSAs. It is automatically updated on a context save operation to point to the next available CSA.

FCX
Free Context List Head Pointer **Reset Value: 0000 0000$_H$**

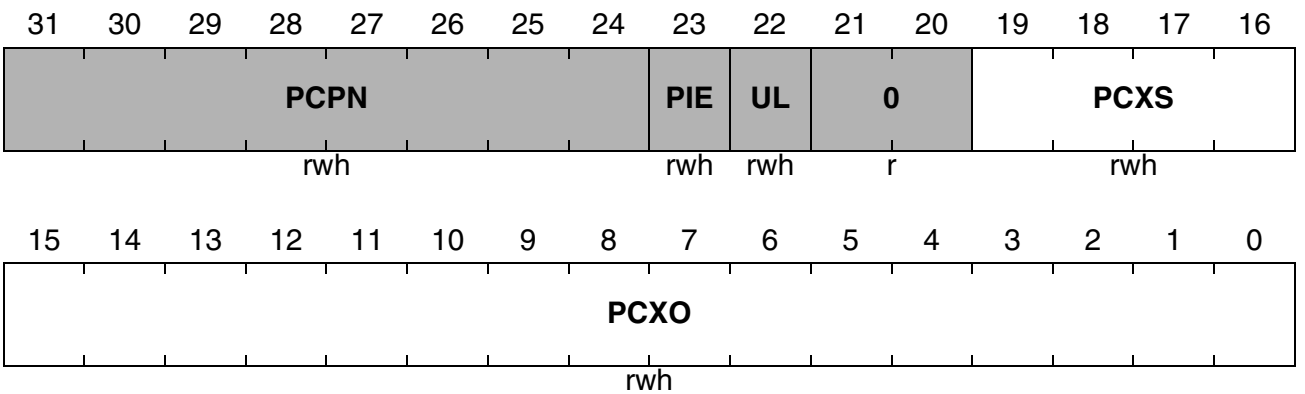


Field	Bits	Type	Description
FCXO	[15:0]	rwh	**FCX Offset Address Field** The combined FCXO and FCXS fields form the FCX pointer, which points to the next available CSA.
FCXS	[19:16]	rwh	**FCX Segment Address Field** This bit field is used in conjunction with the FCXO field.
0	[31:20]	r	**Reserved**; read as 0; should be written with 0;

2.2.2.2 Previous Context Pointer (PCX)

The Previous Context Pointer (PCX) holds the address of the CSA of the previous task. PCX is part of PCXI. It is shown for easy reference. The bits not relevant to the pointer function are shaded.

PCX
Previous Context Pointer **Reset Value: 0000 0000$_H$**



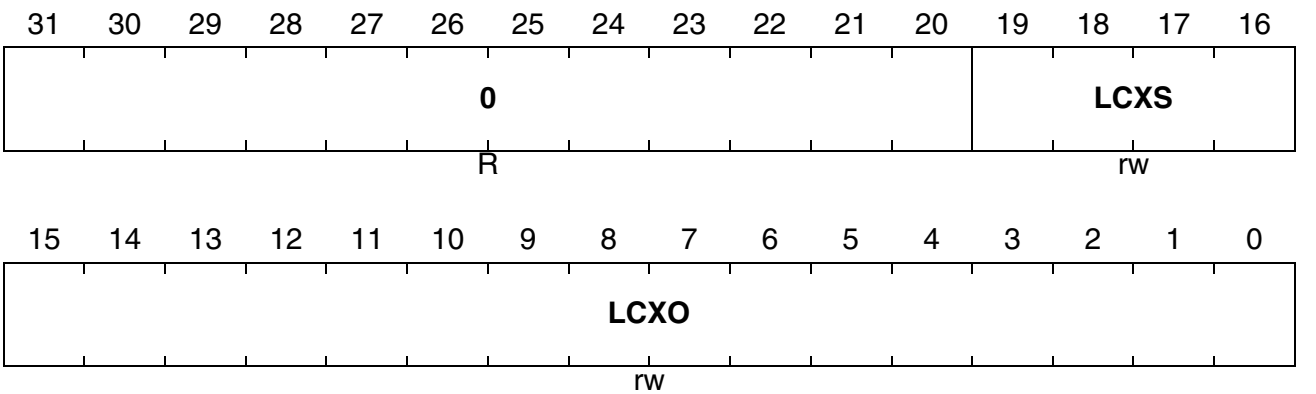
Field	Bits	Type	Description
PCXO	[15:0]	rwh	**Previous Context Pointer Offset Field** The combined PCXO and PCXS fields form the pointer PCX, which points to the CSA of the previous context.
PCXS	[19:16]	rwh	**PCX Segment Address** This field is used in conjunction with the PCXO field-
0	20, 21	r	**Reserved**; read as 0; should be written with 0;

Note: The shaded bit fields are described at register PCXI.

2.2.3 Free Context List Limit Pointer (LCX)

The LCX register points to the last context save area (CSA) in the linked list of free CSAs. The value is used on a context save operation to detect the usage of the last entry, and to trigger a trap to the CPU to allow proper software reaction.

LCX
Free Context List Limit Pointer **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
LCXO	[15:0]	rw	**Previous Context Pointer Offset Field** The LCXO and LCXS fields form the pointer LCX, which points to the last available CSA.
LCXS	[19:16]	rw	**LCX Segment Address** This bit field is used in conjunction with the LCXO field.

2.2.4 Stack Management

General purpose address register A10 is designated as the Stack Pointer (SP). The initial contents of this register are usually set by an RTOS instruction when a task is created. This allows a private stack area to be assigned to individual tasks.

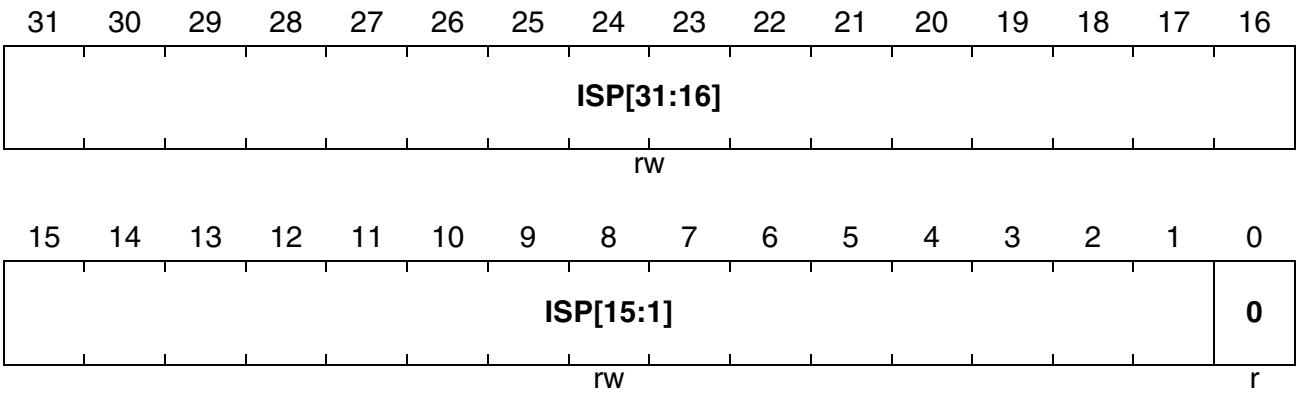
When entering Interrupt Service Routines (ISRs), the Stack Pointer is loaded with the contents of a separate register — the Interrupt Stack Pointer (ISP) — after saving its previous contents with the Upper Context. This helps to prevent interrupt service routines from accessing the private stack areas and possibly interfering with the context of software-managed tasks.

2.2.4.1 Interrupt Stack Pointer (ISP)

To separate the private stack of software managed tasks from the stack used for interrupt service routines (ISRs), an automatic switch is implemented in the TC1765 to use the Interrupt Stack Pointer (ISP) when entering ISRs. After saving the Upper Context, and with it register A10 (used as the stack pointer), register A10 is loaded with the contents of register ISP. When returning from the ISR, the previous value of the Stack Pointer is restored through the Upper Context restore operation.

Note: Register ISP is EndInit-protected!

ISP
Interrupt Stack Pointer **Reset Value: 0000 0100$_H$**



Field	Bits	Type	Description
ISP	[31:1]	rw	**Interrupt Stack Pointer**
0	0	r	**Reserved**; read as 0; should be written with 0;

2.2.5 Interrupt and Trap Control

Three CSFRs support interrupt and trap handling: the Interrupt Control Register (ICR), the Interrupt Vector Table Pointer (BIV), and the Trap Vector Table Pointer (BTV).

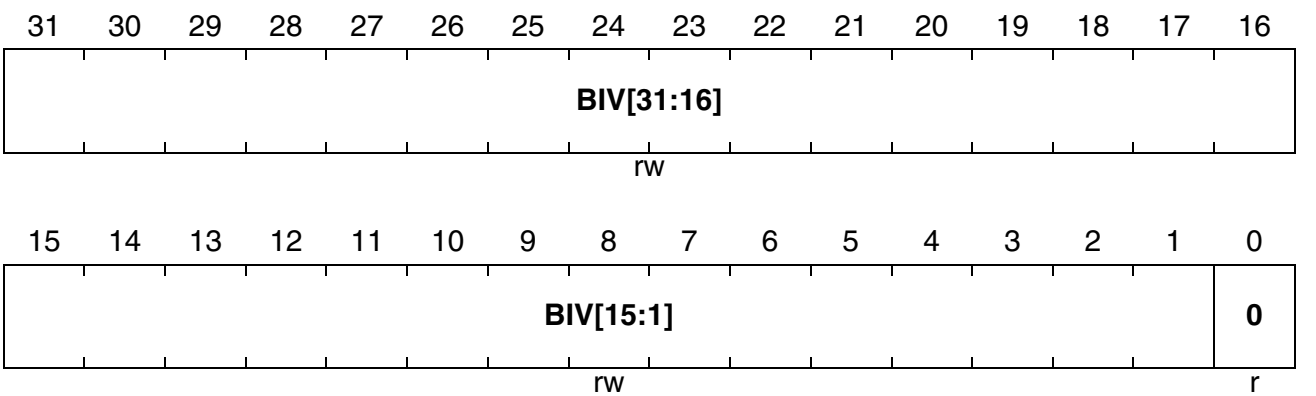
The ICR holds the current CPU priority number (CCPN), the enable/disable bit for the interrupt system, the pending interrupt priority number, and an implementation-specific control for the interrupt arbitration scheme. The other two registers hold the base addresses for the interrupt (BIV) and trap vector tables (BTV).

2.2.5.1 Interrupt Vector Table Pointer (BIV)

The BIV register points to the start address of the Interrupt Vector Table in code memory. More detailed information on the functions associated with this register and the Interrupt Vector Table can be found in **Chapter 14**.

Note: Register BIV is EndInit-protected!

BIV
Interrupt Vector Table Pointer **Reset Value: 0000 0000$_H$**



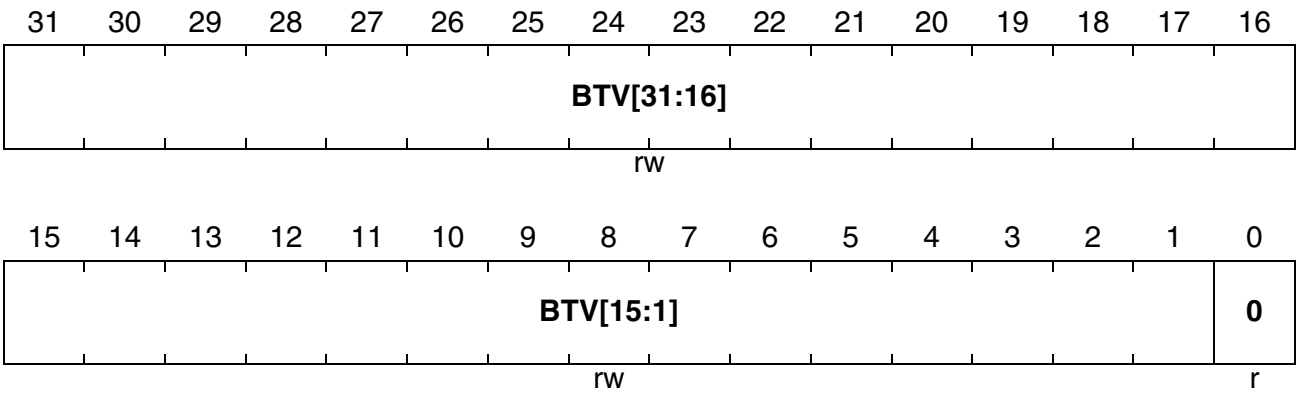
Field	Bits	Type	Description
BIV	[31:1]	rw	**Base Address of Interrupt Vector Table**
0	0	r	**Reserved**; read as 0; should be written with 0;

2.2.5.2 Trap Vector Table Pointer (BTV)

The BTV register points to the start address of the Trap Vector Table in code memory. More detailed information on the functions associated with this register and the Trap Vector Table can be found in **Chapter 15**.

Note: Register BTV is EndInit-protected!

BTV
Trap Vector Table Pointer **Reset Value: A000 0100$_H$**

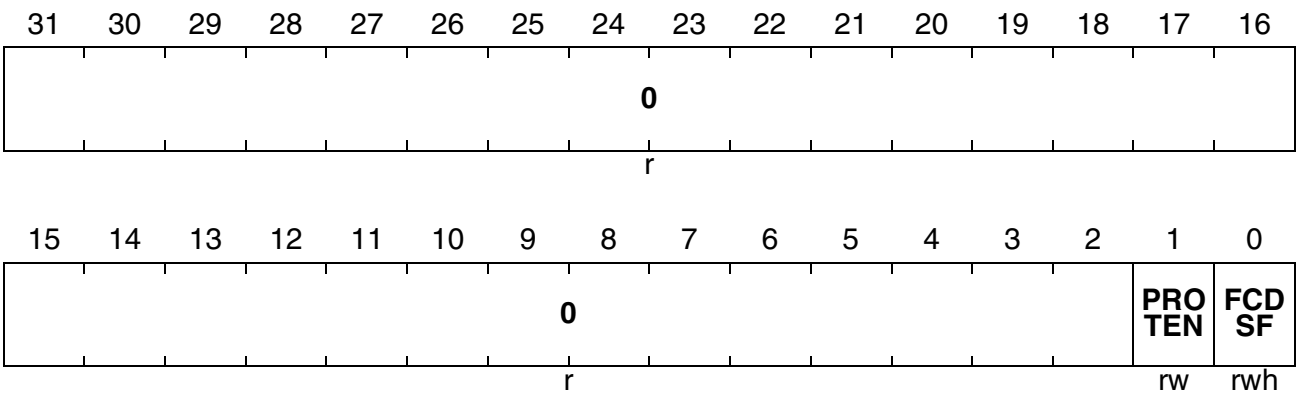


Field	Bits	Type	Description
BTV	[31:1]	rw	**Base Address of Trap Vector Table**
0	0	r	**Reserved**; read as 0; should be written with 0;

2.2.6 System Control Register

The System Configuration Control Register (SYSCON) provides the enable/disable bit for the memory protection system and a status flag for a Free Context List Depletion condition.

SYSCON
System Configuration Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
FCDSF	0	rwh	**Free Context List Depletion Sticky Flag** This sticky bit indicates that a FCD trap occurred since the bit was last cleared by software. 0 No FCD trap occurred since the last clear 1 An FCD trap occurred since the last clear
PROTEN	1	rw	**Memory Protection Enable** PROTEN enables the memory protection system. Memory protection is controlled through the memory protection register sets. Note that it is required to initialize the protection register sets prior to setting PROTEN to 1. 0 Memory Protection is disabled 1 Memory Protection is enabled
0	[31:2]	r	**Reserved**; read as 0; should be written with 0;

2.2.7 Memory Protection Registers

As described in **Section 2.1.11.2**, memory ranges are protected from unauthorized read-, write-, or instruction-fetch accesses. The TC1765 contains register sets (PRSs) that specify the addresses and the access permissions for a number of memory ranges. The TC1765 incorporates two sets each for data and code memory protection. See **Chapter 11** for detailed register descriptions.

2.2.8 Debug Registers

Six registers are implemented in the CPU to support debugging. These registers define the conditions under which a debug event is generated, the actions taken on the assertion of a debug event, and the status information supplied to the debug functions. See **Chapter 19** for detailed register descriptions.

2.2.9 CSFR Address Table

Table 2-2 lists all CSFRs of the TC1765 and their physical addresses. Except for the General Purpose Registers (GPRs), two addresses are given for each of the CSFRs. The 32-bit address represents the mapped address of the register in segment 15. Access to these mapped locations can be performed through the CPU's Slave Interface (CPS) by any FPI Bus master other than the CPU itself. The 16-bit address given for a register is the associated address when performing an access by the CPU through the MTCR and MFCR instructions.

Access modes to the CSFRs are described in the following notes and, therefore, are not contained in **Table 2-2**.

Note: The General Purpose Registers (GPRs) cannot be accessed by the CPU through MTCR and MFCR instructions. Therefore, they do not have a 16-bit address.

Note: Write accesses to CSFRs through the CPS interface by an FPI Bus master while the CPU is running might lead to unexpected behavior. It is strongly advised to write to these registers only when the CPU is halted.

Note: Read and write accesses from the FPI Bus must only be made with word-aligned word accesses. Any access not following this rule will be flagged with a bus error. The read or write operation will not be performed.

Note: Read accesses from the FPI Bus can be performed in User or Supervisor Mode. Write accesses from the FPI Bus must be performed in Supervisor Mode. A write attempt in User Mode will be flagged with a bus error. The write operation will not be performed.

*Note: Registers ISP, BIV, and BTV are EndInit-protected. To write successfully to these registers, the ENDINIT bit in register WDT_CON0 of the Watchdog Timer must be cleared. See **Chapter 18** for detailed information on the EndInit-protection.*

Table 2-2 CSFR Register Table

Register Short Name	Register Long Name	Address
Core Special Function Registers (CSFRs)		
PCXI	Previous Context Information Register	FFFF FE00$_H$
PSW	Program Status Word	FFFF FE04$_H$
PC	Program Counter	FFFF FE08$_H$ / FE08$_H$
SYSCON	System Configuration Register	FFFF FE14$_H$ / FE14$_H$
BIV	Interrupt Vector Table Pointer	FFFF FE20$_H$ / FE20$_H$
BTV	Trap Vector Table Pointer	FFFF FE24$_H$ / FE24$_H$
ISP	Interrupt Stack Pointer	FFFF FE28$_H$ / FE28$_H$
ICR	ICU Interrupt Control Register	FFFF FE2C$_H$ / FE2C$_H$
FCX	Free CSA List Head Pointer	FFFF FE38$_H$ / FE38$_H$
LCX	Free CSA List Limit Pointer	FFFF FE3C$_H$ / FE3C$_H$
General Purpose Registers (GPRs)		
D0	Data Register D0 (DGPR)	FFFF FF00$_H$
D1	Data Register D1 (DGPR)	FFFF FF04$_H$
D2	Data Register D2 (DGPR)	FFFF FF08$_H$
D3	Data Register D3 (DGPR)	FFFF FF0C$_H$
D4	Data Register D4 (DGPR)	FFFF FF10$_H$
D5	Data Register D5 (DGPR)	FFFF FF14$_H$
D6	Data Register D6 (DGPR)	FFFF FF18$_H$
D7	Data Register D7 (DGPR)	FFFF FF1C$_H$
D8	Data Register D8 (DGPR)	FFFF FF20$_H$
D9	Data Register D9 (DGPR)	FFFF FF24$_H$
D10	Data Register 10 (DGPR)	FFFF FF28$_H$
D11	Data Register 11 (DGPR)	FFFF FF2C$_H$
D12	Data Register 12 (DGPR)	FFFF FF30$_H$
D13	Data Register 13 (DGPR)	FFFF FF34$_H$
D14	Data Register 14 (DGPR)	FFFF FF38$_H$
D15	Data Register 15 (DGPR)	FFFF FF3C$_H$
A0	Address Register 0 (AGPR) Global Address Register	FFFF FF80$_H$

Table 2-2 CSFR Register Table (cont'd)

Register Short Name	Register Long Name	Address
A1	Address Register 1 (AGPR) Global Address Register	FFFF FF84$_H$
A2	Address Register 2 (AGPR)	FFFF FF88$_H$
A3	Address Register 3 (AGPR)	FFFF FF8C$_H$
A4	Address Register 4 (AGPR)	FFFF FF90$_H$
A5	Address Register 5 (AGPR)	FFFF FF94$_H$
A6	Address Register 6 (AGPR)	FFFF FF98$_H$
A7	Address Register 7 (AGPR)	FFFF FF9C$_H$
A8	Address Register 8 (AGPR) Global Address Register	FFFF FFA0$_H$
A9	Address Register 9 (AGPR) Global Address Register	FFFF FFA4$_H$
A10 (SP)	Address Register 10 (AGPR) Stack Pointer	FFFF FFA8$_H$
A11 (RA)	Address Register 11 (AGPR) Return Address	FFFF FFAC$_H$
A12	Address Register 12 (AGPR)	FFFF FFB0$_H$
A13	Address Register 13 (AGPR)	FFFF FFB4$_H$
A14	Address Register 14 (AGPR)	FFFF FFB8$_H$
A15	Address Register 15 (AGPR)	FFFF FFBC$_H$

2.3 Instruction Set Overview

This section provides an overview of the TriCore instruction set architecture. The basic properties and uses of each instruction type are described, as well as the selection and use of the 16-bit (short) instructions.

Note: The "TriCore Architecture Manual" describes each instruction more detailed.

2.3.1 Arithmetic Instructions

Arithmetic instructions operate on data and addresses in registers. Status information about the result of the arithmetic operations is recorded in the five status flags in the Program Status Word (PSW) register. The status flags are described in **Table 2-3**.

Table 2-3 PSW Status Flags

Status Flag	Description
C	**Carry Flag** This flag is set as the result of a carry out from an addition or subtraction instruction. Carry out can result from either signed or unsigned operations. It is also set by arithmetic shift.
V	**Overflow Flag** This flag is set when the signed result cannot be represented in the data size of the result; for example, when the result of a signed 32-bit operation is greater than 2^{31} - 1.
SV	**Sticky Overflow Flag** This flag is set when the overflow flag is set. It remains set until it is explicitly cleared by an RSTV (Reset Overflow bits) instruction.
AV	**Advance Overflow Flag** This flag is updated by all instructions that update the overflow flag and no others. This flag is determined as the Boolean exclusive-or of the two most-significant bits of the result.
SAV	**Sticky Advance Overflow Flag** This flag is set whenever the advanced overflow flag is set. It remains set until it is explicitly cleared by an RSTV (Reset Overflow bits) instruction.

The two signed overflow conditions (overflow and advance overflow) are calculated for all arithmetic instructions. In the case of packed instructions, the conditions are the OR of the conditions for each byte or half-word (parallel) operation. In the case of the multiply-accumulate instructions, the conditions are calculated after the accumulate operation. The unsigned overflow condition is carry for addition or borrow (no carry) for subtraction.

Numerically, overflow for signed 32-bit values occurs when a positive result is greater than $7FFFFFFF_H$ or a negative result is smaller than 80000000_H. Overflow for unsigned 32-bit values occurs when the result is greater than $FFFFFFFF_H$ or less than 00000000_H.

The status flags can be read by software using the Move From Core Register (MFCR) instruction and can be written using the Move to Core Register (MTCR) instruction. The Trap on Overflow (TRAPV) and Trap on Sticky Overflow (TRAPSV) instructions can be used to cause a trap if the V and SV bits, respectively, are set. The overflow bits can be cleared using the Reset Overflow Bits instruction (RSTV).

Individual arithmetic operations can be checked for overflow by reading and testing V. If it is necessary to know only if an overflow occurred somewhere in an entire block of computation, then the SV bit is reset before the block (using the RSTV instruction) and is tested after completion of the block (using MFCR). Jumping based on the overflow result can be done using a MFCR followed by a JZ.T or JNZ.T (conditional jump on the value of a bit).

The AV and SAV bits are set as a result of the exclusive OR of the two most-significant bits of the particular data type (byte, half-word, word, or double-word) of the result, which indicates that an overflow almost occurred.

Because most signal-processing applications can handle overflow by simply saturating the result, most of the arithmetic instructions have a saturating version for signed and unsigned overflow. Note that saturating versions of all instructions can be synthesized using short code sequences.

When saturation is used for 32-bit signed arithmetic overflow, if the true result of the computation is greater than $(2^{31} - 1)$ or less than -2^{31}, the result is set to $(2^{31} - 1)$ or -2^{31}, respectively. The bounds for 16-bit signed arithmetic are $(2^{15} - 1)$ and -2^{15}. The bounds for 8-bit signed arithmetic are $(2^7 - 1)$ and -2^7. When saturation is used for unsigned arithmetic, the lower bound is always zero and the upper bounds are $(2^{32} - 1)$, $(2^{16} - 1)$, and $(2^8 - 1)$. Saturation is indicated in the instruction mnemonic by an “S”, and unsigned is indicated by a “U” following the period (.). For example, the instruction mnemonic for a signed saturating addition is ADDS, and the mnemonic for an unsigned saturating addition is ADDS.U. Saturation is also used for signed fractions in DSP operations.

2.3.1.1 Integer Arithmetic

Move

Move instructions move a value in a data register or a constant value in the instruction to a destination data register. Move can be used to quickly load a large constant into a data register. A 16-bit constant is created using MOV (which sign-extends the value to 32 bits) or MOV.U (which zero-extends to 32 bits). The MOVH (Move Highword) instruction loads a 16-bit constant into the most-significant sixteen bits of the register and zero fills the least significant sixteen bits, which is useful for loading a left-justified

constant fraction. Loading a 32-bit constant can be done using a MOVH instruction followed by an ADDI (Add Immediate), or by a MOV.U followed by ADDIH (Add Immediate High Word).

Addition and Subtraction

There are three types of addition instructions: no saturation (ADD), signed saturation (ADDS), and unsigned saturation (ADDS.U). For extended precision addition, the ADDX (Add Extended) instruction sets the PSW carry bit to the value of the ALU carry out. The ADDC (Add with Carry) instruction uses the PSW carry bit as the carry in, and updates the PSW carry bit with the ALU carry out. For extended precision addition, the least significant word of the operands is added using the ADDX instruction, and the remaining words are added using the ADDC instruction. The ADDC and ADDX instructions do not support saturation.

Often it is necessary to add 16-bit or 32-bit constants to integers. The ADDI (Add Immediate) and ADDIH (Add Immediate High) instructions add a 16-bit, sign-extended constant or a 16-bit constant, left-shifted by 16. Addition of any 32-bit constant can be done using ADDI followed by an ADDIH.

All add instructions except those with constants have similar corresponding subtract instructions. Because the large immediate of ADDI is sign-extended, it may be used for both addition and subtraction.

The RSUB (Reverse Subtract) instruction subtracts a register from a constant. Using zero as the constant yields negation as a special case.

Multiply and Multiply-Add

Multiplication of two 32-bit integers that produce a 32-bit result can be handled using MUL (Multiply Signed), MULS (Multiply Signed with Saturation), and MULS.U (Multiply Unsigned with Saturation). The MULM (Multiply with Multiword Result) and MULM.U (Multiply with Multiword Result Unsigned) instructions produce the full 64-bit result, which is stored to a register pair; MULM is for signed integers, and MULM.U is for unsigned integers. Special multiply instructions are used for DSP operations.

The Multiply-Add instruction (MADD) multiplies two signed operands, adds the result to a third operand, and stores the result in a destination. Because the third operand and the destination do not use the same registers, the intermediate sums of a multi-term multiply-add instruction can be saved without requiring any additional register moves. The MADD, MADDS (Multiply-Add with Saturation), and MADDS.U (Multiply-Add with Saturation Unsigned) instructions operate on and produce 32-bit integers; MADDS and MADDS.U will saturate on signed and unsigned overflow, respectively. The instructions MADDM (Multiply-Add with Multiword Result), MADDM.U (Multiply-Add with Multiword Result Unsigned), MADDMS (Multiply-Add Multiword with Saturation), and MADDMS.U (Multiply-Add Multiword with Saturation Unsigned) can be used to add the 64-bit product to a 64-bit source and produce a 64-bit result.

The set of Multiply-Subtract (MSUB) instructions that supports the accumulation of products using subtraction instead of addition provides the same set of variations as the MADD instructions.

Division

Division of 32-bit by 32-bit integers is supported for both signed and unsigned integers. Because an atomic divide instruction would require an excessive number of cycles to execute, a divide-step sequence is used to reduce interrupt latency. The divide step sequence allows the divide time to be proportional to the number of significant quotient bits expected.

The sequence begins with a Divide-Initialize instruction (DVINIT(.U), DVINIT.H(U), or DVINIT.B(U), depending on the size of the quotient and whether the operands are to be treated as signed or unsigned). The divide initialization instruction extends the 32-bit dividend to 64 bits, then shifts it left by 0, 16, or 24 bits. Simultaneously it shifts in that many copies of the quotient sign bit to the low-order bit positions. Then follows 4, 2, or 1 Divide-Step instructions (DVSTEP or DVSTEP.U). Each divide step instruction develops eight bits of quotient.

At the end of the divide step sequence, the 32-bit quotient occupies the low-order word of the 64-bit dividend register pair and the remainder is held in the high-order word. If the divide operation was signed, the Divide-Adjust instruction (DVADJ) is required to perform a final adjustment of negative values. If the dividend and the divisor are both known to be positive, the DVADJ instruction can be omitted.

Absolute Value, Absolute Difference

A common operation on data is the computation of the absolute value of a signed number or the absolute value of the difference between two signed numbers. These operations are provided directly by the ABS and ABSDIF instructions and there is a version of each instruction which saturates when the result is too large to be represented as a signed number.

Min, Max, Saturate

Instructions are provided that directly calculate the minimum or maximum of two operands. The MIN and MAX instructions are used for signed integers, MIN.U and MAX.U are used for unsigned integers. The SAT instructions can be used to saturate the result of a 32-bit calculation before storing it in a byte or half-word in memory or a register.

Conditional Arithmetic Instructions

The conditional instructions — Conditional Add (CADD), Conditional Subtract (CSUB), and Select (SEL) — provide efficient alternatives to conditional jumps around very short

sequences of code. All of the conditional instructions use a condition operand that controls the execution of the instruction. The condition operand is a data register with any non-zero value interpreted as TRUE and a zero value interpreted as FALSE. For the CADD and CSUB instructions, the addition/subtraction is performed if the condition is TRUE. For the CADDN and CSUBN instructions it is performed if the condition is FALSE.

The SEL instruction copies one of its two source operands to its destination operand, with the selection of source operands determined by the value of the condition operand (This operation is the same as the C language “?” operation). A typical use might be to record the index value yielding the larger of two array elements:

```
index_max = (a[i] > a[j]) ? i : j;
```

If one of the two source operands in a Select instruction is the same as the destination operand, then the Select instruction implements a simple conditional move. This occurs fairly often in source statements of the general form:

```
if (<condition>) then <variable> = <expression>;
```

Provided that `<expression>` is simple, it is more efficient to evaluate it unconditionally into a source register, using a SEL instruction to perform the conditional assignment, rather than conditionally jumping around the assignment statement.

Logical

The TriCore architecture provides a complete set of 2-operand, bit-wise logic operations. In addition to the AND, OR, and XOR functions, there are the negations of the output — NAND, NOR, and XNOR — and negations of one of the inputs — ANDN and ORN (the negation of an input for XOR is the same as XNOR).

Count Leading Zeroes, Ones, and Signs

To provide efficient support for normalization of numerical results, prioritization, and certain graphics operations, three Count Leading instructions are provided: CLZ (Count Leading Zeros), CLO (Count Leading Ones), and CLS (Count Leading Signs). These instructions are used to determine the amount of left shifting necessary to remove redundant zeros, ones, or signs.

Note that the CLS instruction returns the number of leading redundant signs, which is the number of leading signs minus one. Furthermore, the following special cases are defined: CLZ(0) = 32, CLO(-1) = 32, and CLS(0) = CLS(-1) = 31.

For example, CLZ returns the number of consecutive zeros starting from the most-significant bit of the value in the source data register. In the example shown below (**Table 2-8**), there are seven zeros in the most-significant portion of the input register. If the most-significant bit of the input is a 1, CLZ returns 0.

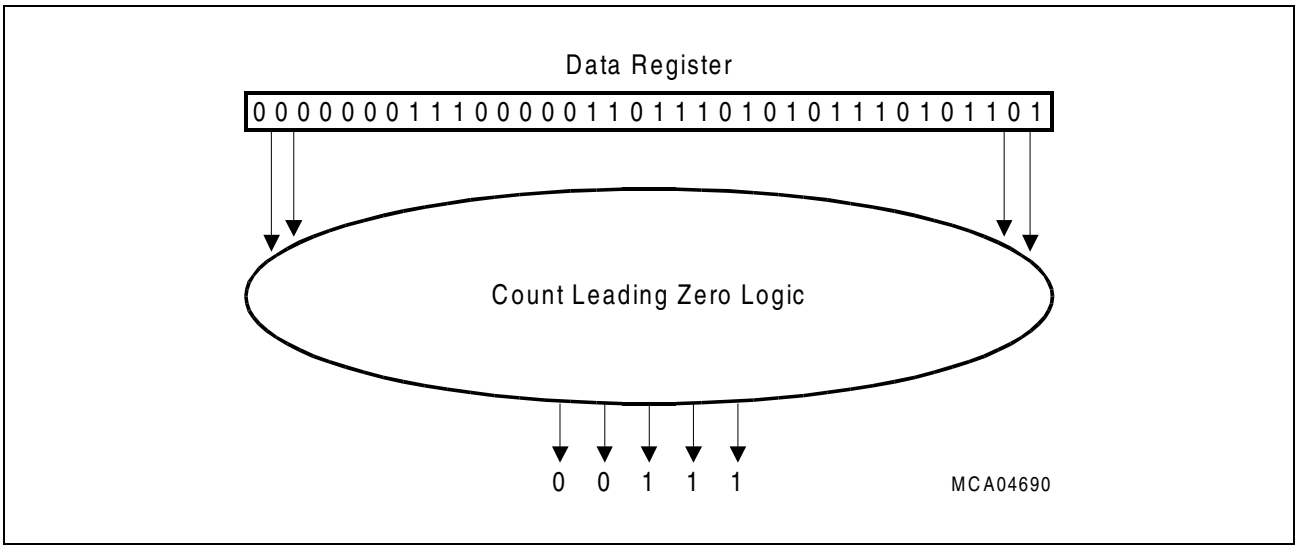


Figure 2-8 Operation of CLZ Instruction

The Count Leading instructions are useful for parsing certain Huffman codes and bit strings consisting of Boolean flags because the code or bit string can be quickly classified by determining the position of the first one (scanning from left to right).

Shift

The shift instructions support multi-bit shifts. The shift amount is specified by a signed integer (n), which may be the contents of a register or a sign-extended constant in the instruction. If n ≥ 0, the data is shifted left by n[4:0]; otherwise, the data is shifted right by (-n)[4:0]. The (logical) shift instruction, SH, shifts in zeroes for both right and left shifts; the arithmetic shift instruction, SHA, shifts in sign bits for right shifts and zeroes for left shifts. The arithmetic shift with saturation instruction, SHAS, will saturate (on a left shift) if the sign bits that are shifted out are not identical to the sign bit of the result.

Bit Field Extract and Insert

The TriCore architecture supports three bit field extract instructions. The EXTR.U and EXTR instructions extract w (width) consecutive bits from the source, beginning with the bit number specified by the pos (position) operand. The width and position can be specified by two immediate values, by an immediate value and a data register, or by a data register pair. The EXTR.U instruction (**Figure 2-9**) zero-fills the most significant (32-w) bits of the result.

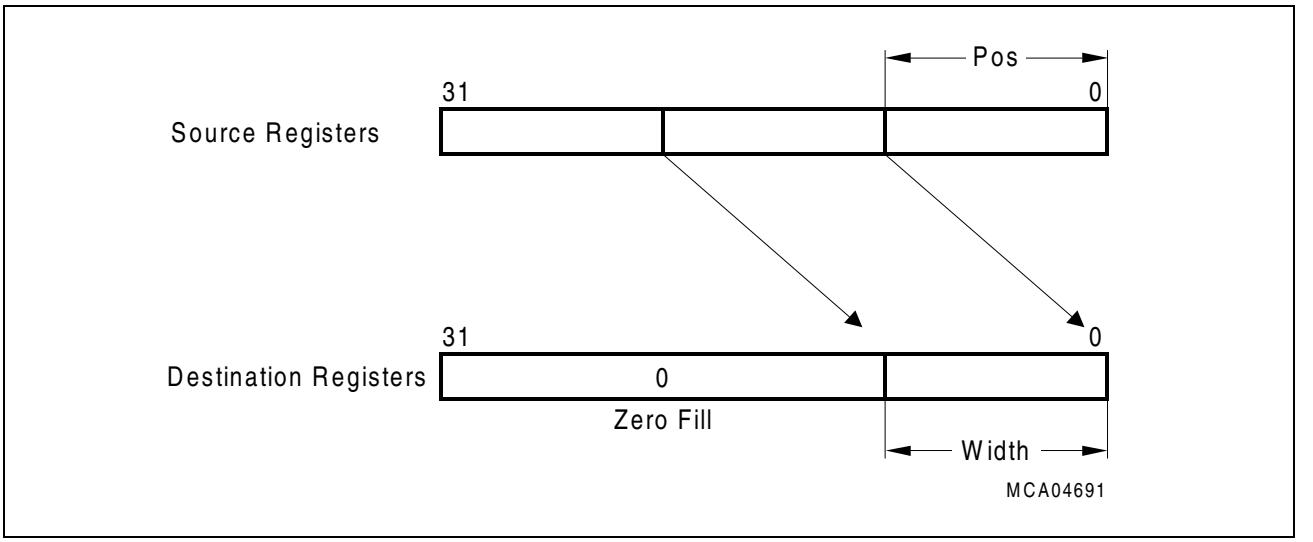


Figure 2-9 Operation of EXTR.U Instruction

The EXTR instruction (**Figure 2-10**) fills the most-significant bits of the result by sign-extending the bit field extracted (thus duplicating the most-significant bit of the bit field).

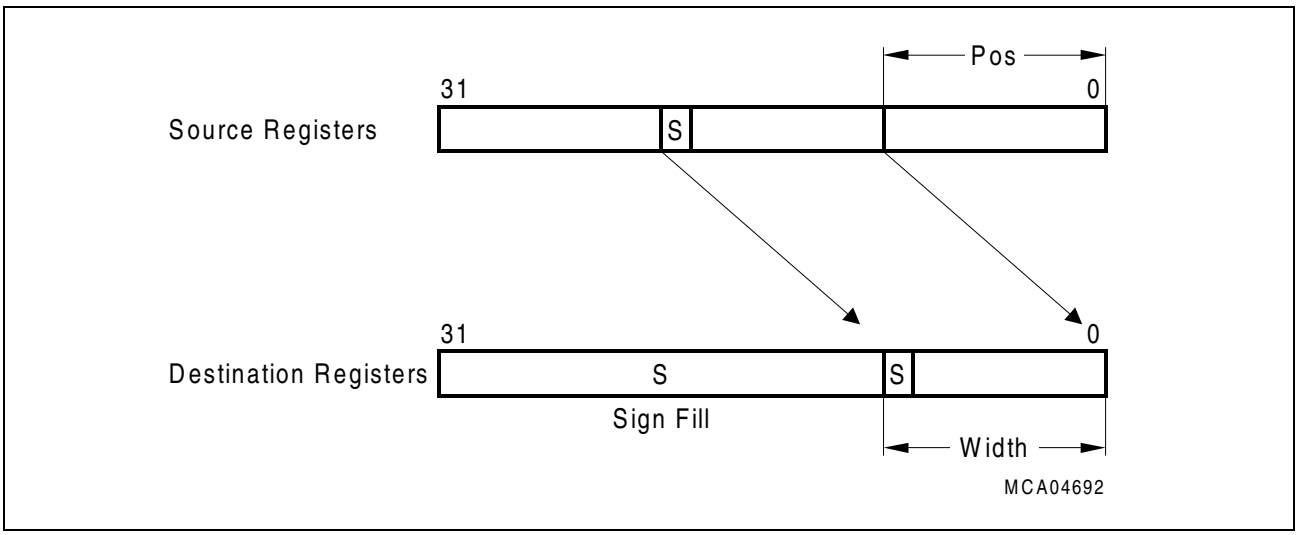


Figure 2-10 Operation of EXTR Instruction

The DEXTR instruction (**Figure 2-11**), concatenates two data register sources to form a 64-bit value from which 32 consecutive bits are extracted. The operation can be thought of as a left shift by pos bits, followed by the truncation of the least significant 32 bits of the result. The value of pos is contained in a data register or is an immediate value in the instruction.

The DEXTR instruction can be used to normalize the result of a DSP filter accumulation in which a 64-bit accumulator is used with several guard bits. The value of pos can be determined by using the CLS (Count Leading Signs) instruction. The DEXTR instruction can also be used to perform a multi-bit rotation by using the same source register for both of the sources that are concatenated.

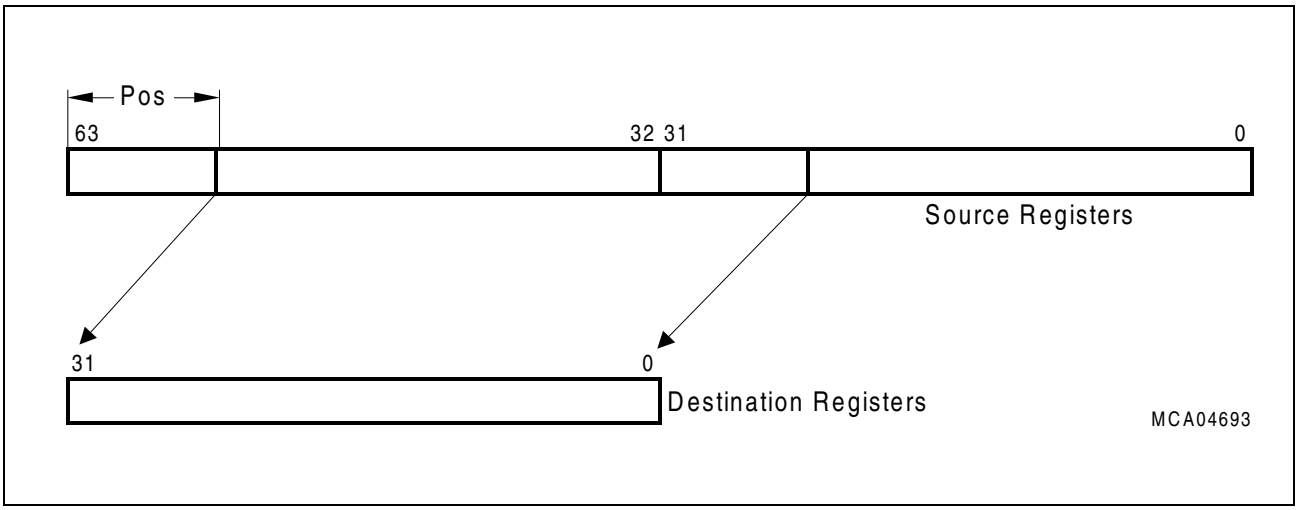


Figure 2-11 Operation of DEXTR Instruction

The INSERT instruction (**Figure 2-12**) takes the w least significant bits of a source data register, shifted left by pos bits and substitutes them into the value of another source register. All other (32-w) bits of the value of the second register are passed through. The values of width and pos are specified in the same way as for EXTR(.U). There is also an alternative form of INSERT that allows a zero-extended 4-bit constant to be the value which is inserted.

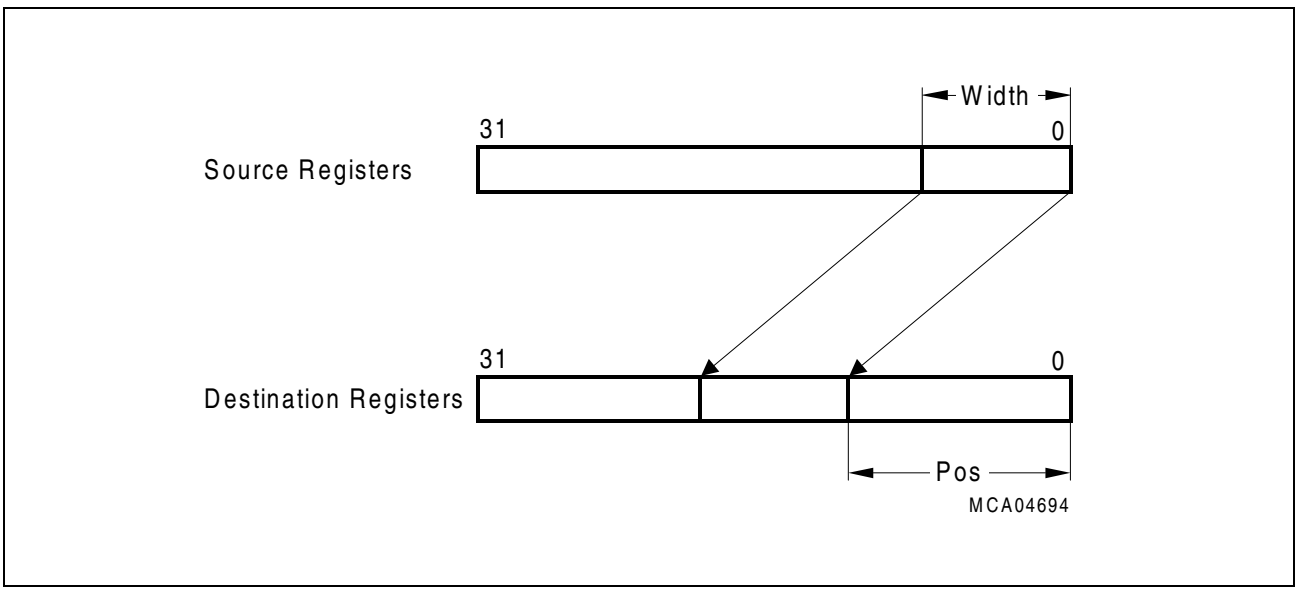


Figure 2-12 Operation of INSERT Instruction

2.3.1.2 DSP Arithmetic

DSP arithmetic instructions operate on 16-bit, signed fractional data in the 1.15 format (also known as Q15) and 32-bit signed fractional data in 1.31 format (also known as Q31). Data values in this format have a single high-order sign bit with a value of 0 or -1, followed by an implied binary point and fraction. Their values are in the range [-1, 1].

16-bit DSP data is loaded into the most significant half of a data register, with the 16 least significant bits set to zero. The left alignment of 16-bit data allows it to be added directly to 32-bit data in 1.31 format. All other fractional formats can be synthesized by explicitly shifting data as required.

Operations created for this format are multiplication, multiply-add, and multiply-subtract. The signed fractional formats 1.15 and 1.31 are supported with the MUL.Q and MULR.Q instructions. These instructions operate on two left-justified signed fractions and return a 32-bit signed fraction.

Scaling

The multiplier result can be shifted in two ways:

- Left shifted by 1
  - One sign bit is suppressed and the result is left-aligned, conserves the input format.
- Not shifted
  - The result retains its two sign bits (2.30 format).
  - This format can be used with IIR filters, in which some of the coefficients are between 1 and 2, and to have one guard bit for accumulation.

Special Case = -1 × -1 = +1

When multiplying the two maximum negative values (-1), the result should be the maximum positive number (+1). For example,

```
0x8000 * 0x8000 = 0x4000 0000
```

is correctly interpreted in Q format as:

```
-1(1.15 format) * -1(1.15 format) = +1 (2.30 format)
```

However, when the result is shifted left by one, the result is 0x8000 0000, which is incorrectly interpreted as:

```
-1(1.15 format) * -1(1.15 format) = -1 (1.31 format)
```

To avoid this problem, the result of a Q format operation (-1 * -1) that has been left-shifted by one (left-justified), is saturated to the maximum positive value. Thus,

```
0x8000 * 0x8000 = 0x7FFF FFFF
```

is correctly interpreted in Q format as:

```
-1(1.15 format) * -1(1.15 format) = (nearest representation of)+1 (1.31
format)
```

This operation is completely transparent to the user and does not set the overflow flags.

Guard Bits

When accumulating sums (for example, in filter calculations) guard bits are often required to prevent overflow. The instruction set directly supports the use of one guard bit when using a 32-bit accumulator. When more guard bits are required, a register pair (64 bits) can be used.

Rounding

Rounding is used to retain the 16-bit most-significant bits of a 32-bit result. Rounding is combined with the MUL, MADD, MSUB instructions, and is implemented by adding 1 to bit 15 of a 32-bit register.

Overflow and Saturation

Saturation on signed and unsigned overflow is implemented as part of the MUL, MADD, and MSUB instructions.

Sticky Advance Overflow and Block Scaling in FFT

The Sticky Advance Overflow (SAV) bit is set whenever an overflow “almost” occurs. It can be used in block scaling of intermediate results during an FFT calculation. Before each pass of applying a butterfly operation, the SAV bit is cleared, and after the pass the SAV bit is tested. If it is set, all of the data is scaled (using an arithmetic right shift) before starting the next pass. This procedure gives the greatest dynamic range for intermediate results without the risk of overflow.

Packed Arithmetic

The packed arithmetic instructions partition a 32-bit word into several identical objects, which can then be fetched, stored, and operated on in parallel. These instructions, in particular, allow the full exploitation of the 32-bit word of the TriCore architecture in signal and data processing applications.

The TriCore architecture supports two packed formats. The first format (**Figure 2-13**) divides the 32-bit word into two, 16-bit (half-word) values. Instructions which operate on data in this way are denoted in the instruction mnemonic by the “.H” and “.HU” data type modifiers.

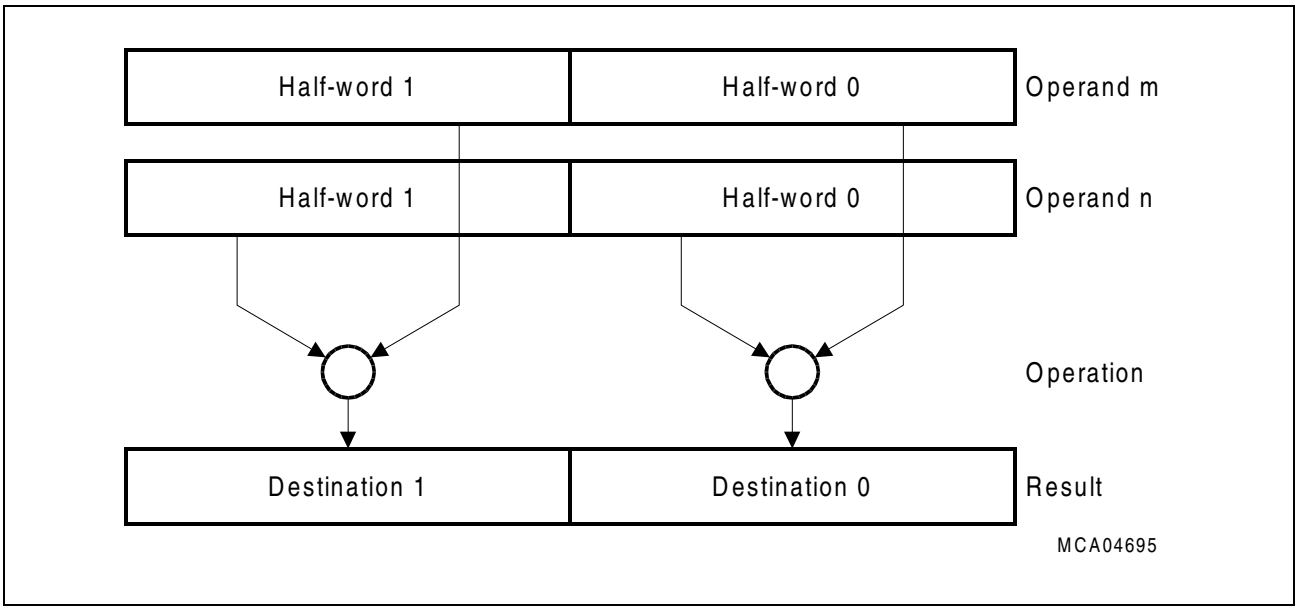


Figure 2-13 Packed Half-word Data Format

The second packed format (**Figure 2-14**) divides the 32-bit word into four, 8-bit values. Instructions that operate this way are denoted by the ".B" and ".BU" data type modifiers.

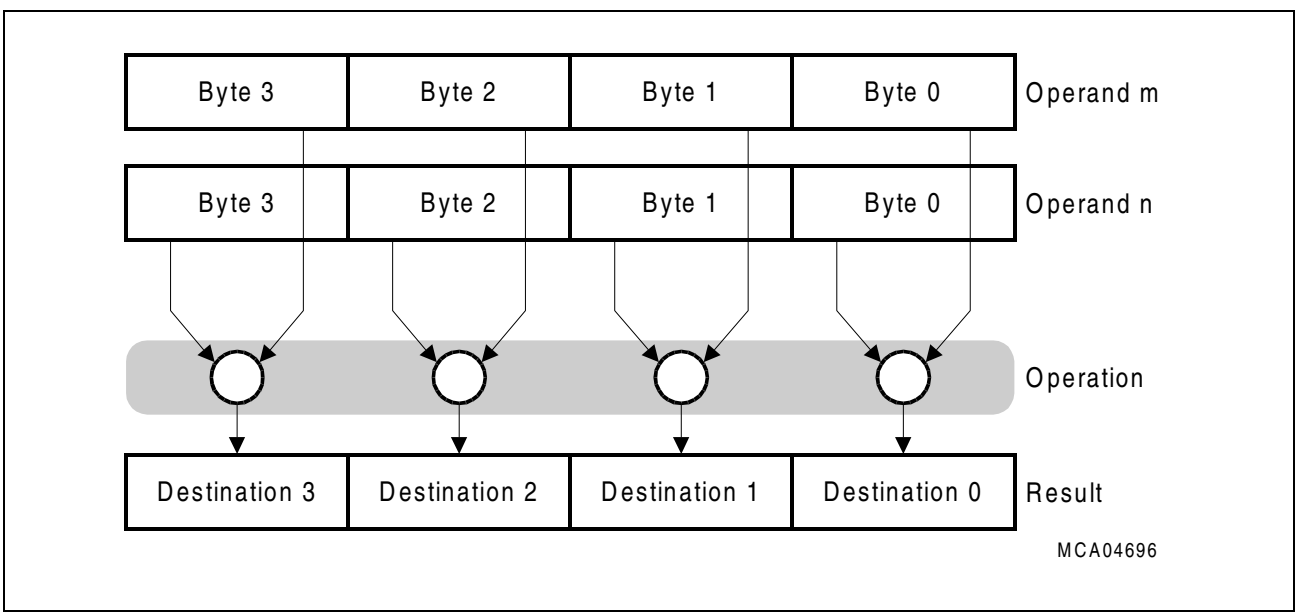


Figure 2-14 Packed Byte Data Format

The loading and storing of packed values into data registers is supported by the normal Load Word (LD.W) and Store Word (ST.W) instructions. The packed objects can then be manipulated in parallel by a set of special packed arithmetic instructions that perform such arithmetic operations as addition, subtraction, multiplication, etc.

Addition is performed on individual packed bytes or half-words using the ADD.B and ADD.H instructions and their saturating variations ADDS.B and ADDS.H. ADD.B ignores

overflow/underflow within individual bytes, while ADDS.B will saturate individual bytes to the most positive, 8-bit signed integer (127) on individual overflow, or to the most negative, 8-bit signed integer (-128) on individual underflow. Similarly, the ADD.H instruction ignores overflow/underflow within individual half-words, while the ADDS.H will saturate individual half-words to the most positive 16-bit signed integer (2^{15} - 1) on individual overflow, or to the most negative 16-bit signed integer (-2^{15}) on individual underflow. Saturation for unsigned integers is also supported by the ADDS.BU and ADDS.HU instructions. Arithmetic on packed data also includes subtraction, multiplication, absolute value, and absolute difference.

2.3.2 Compare Instructions

The compare instructions use a perform operation on the contents of two registers. The Boolean result (1 = true and 0 = false) is stored in the least significant bit of a data register, and the remaining bits in the register are cleared to zero. **Figure 2-15** illustrates the operation of the LT (Less Than) compare instruction.

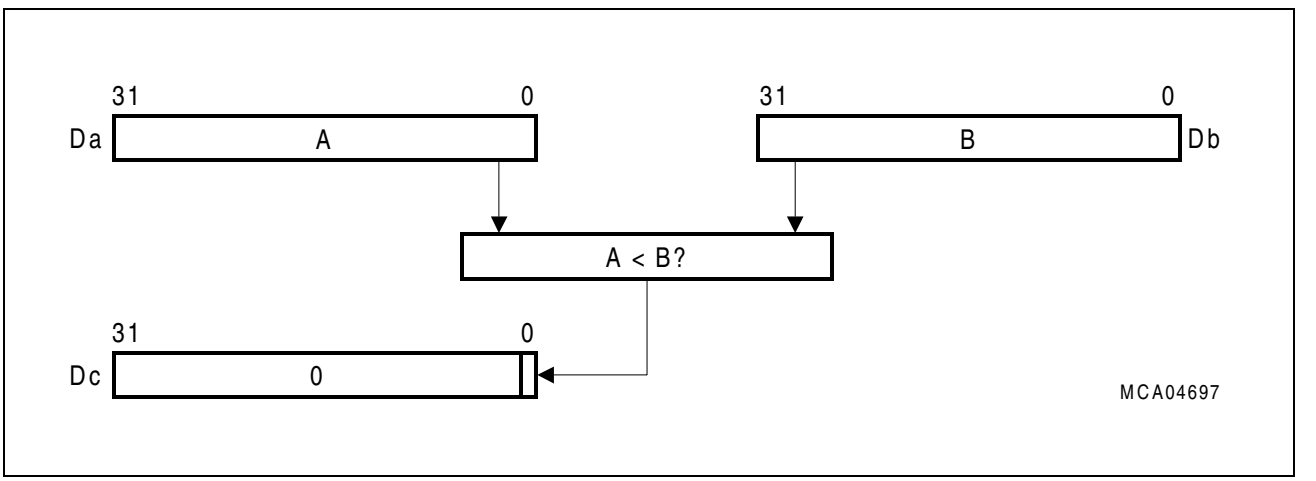


Figure 2-15 LT Comparison

The comparison instructions are: equal (EQ), not equal (NE), less than (LT), and greater than or equal to (GE), with versions for both signed and unsigned integers.

Comparison conditions not explicitly provided in the instruction set can be obtained by either swapping the operands when comparing two registers, or by incrementing the constant by one when comparing a register and a constant (**Table 2-4**).

Table 2-4 Equivalent Comparison Operations

"Missing" Comparison Operation	TriCore Equivalent Comparison Operation
LE Dc, Da, Db	GE Dc, Db, Da
LE Dc, Da, const	LT Dc, Da, (const + 1)
GT Dc, Da, Db	LT Dc, Db, Da
GT Dc, Da, const	GE Dc, Da, (const + 1)

To accelerate the computation of complex conditional expressions, the accumulation of versions of the comparison instructions are supported. These instructions — as indicated in the instruction mnemonic by "op" preceding the "." (for example, op.LT) — combine the result of the comparison with a previous comparison result. The combination is a logic AND, OR, or XOR; for example, AND.LT, OR.LT, and XOR.LT. **Figure 2-16** illustrates combining the LT instruction with a Boolean operation.

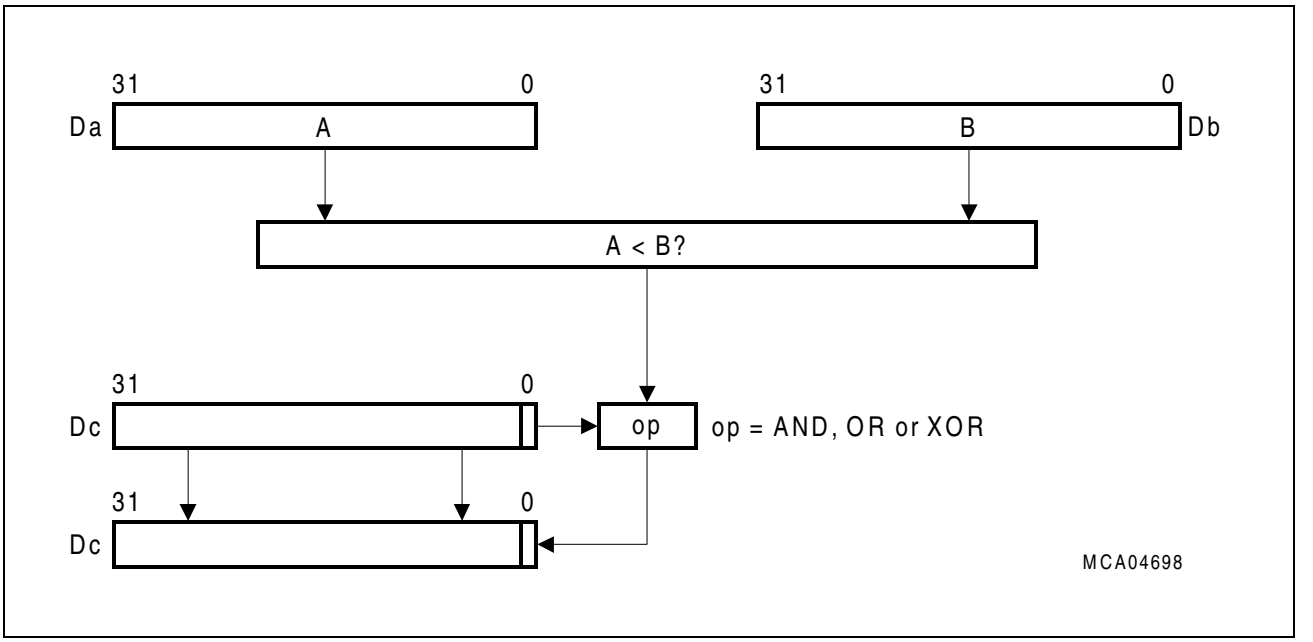


Figure 2-16 Combining LT Comparison with Boolean Operation

The evaluation of the following C expression can be optimized using the combined compare-Boolean operation:

```
d5 = (d1 < d2) || (d3 == d4);
```

Assuming all variables are in registers, two instructions will compute the value in d5:

```
lt      d5,d1,d2     ; compute (d1 < d2)
or.eq   d5,d3,d4     ; or with (d3 == d4)
```

Certain control applications require that several Booleans be packed into a single register. These packed bits can be used as an index into a table of constants or a jump table, which permits complex Boolean functions and/or state machines to be evaluated efficiently. To facilitate the packing of Boolean results into a register, compound Compare with Shift instructions (for example, SH.EQ) are supported. The result of the comparison is placed in the least significant bit of the result after the contents of the destination register have been shifted left by one position. **Figure 2-17** illustrates the operation of the SH.LT (Shift Less Than) instruction.

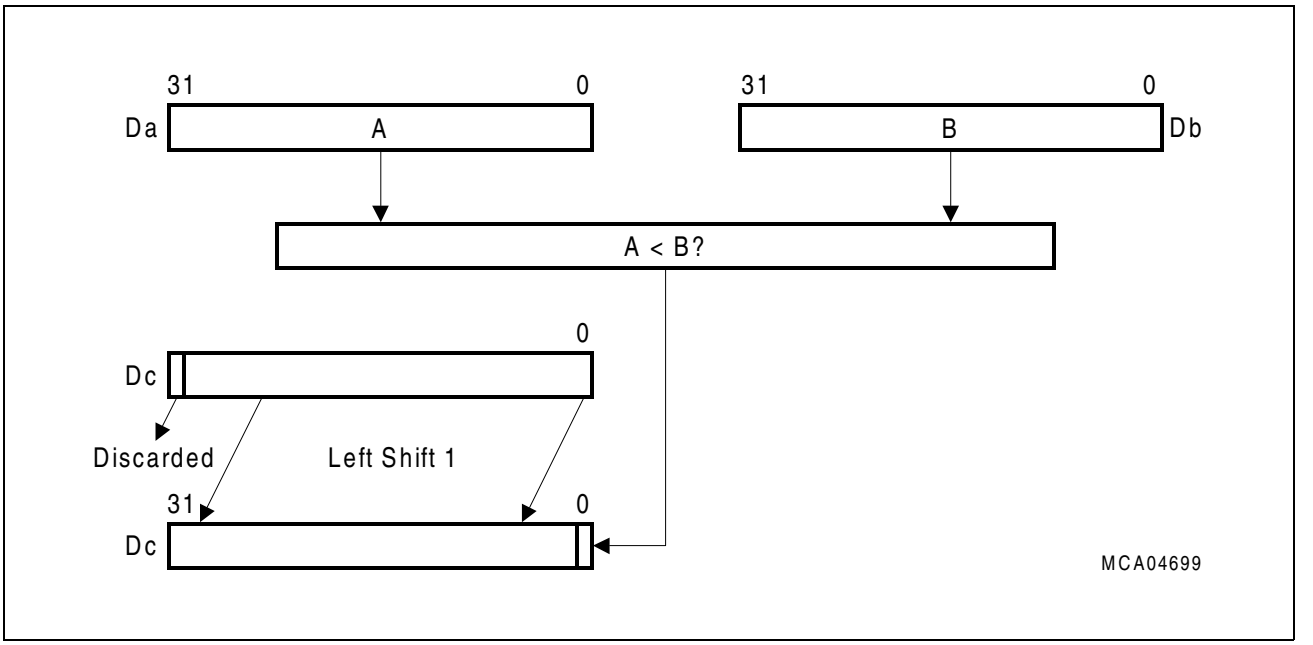


Figure 2-17 SH.LT Instruction

For packed bytes, there are special compare instructions that perform four individual byte comparisons and produce a 32-bit mask consisting of four “extended” Booleans. For example, EQ.B yields a result where individual bytes are FF_H for a match or 00_H for no match. Similarly, for packed half-words there are special compare instructions that perform two individual half-word comparisons and produce two extended Booleans. The EQ.H instruction results in two extended Booleans: $FFFF_H$ for a match and 0000_H for no match. There are even abnormal packed-word compare instructions that compare two words in the normal way but produce a single extended Boolean. The EQ.W instruction results in the extended Boolean $FFFFFFFF_H$ for match and 00000000_H for no match.

Extended Booleans are useful as masks, that can be used by subsequent bit-wise logic operations. Also, CLZ (count leading zeros) or CLO (count leading ones) can be used on the result to quickly find the position of the left-most match. **Figure 2-18** shows an example of the EQ.B instruction.

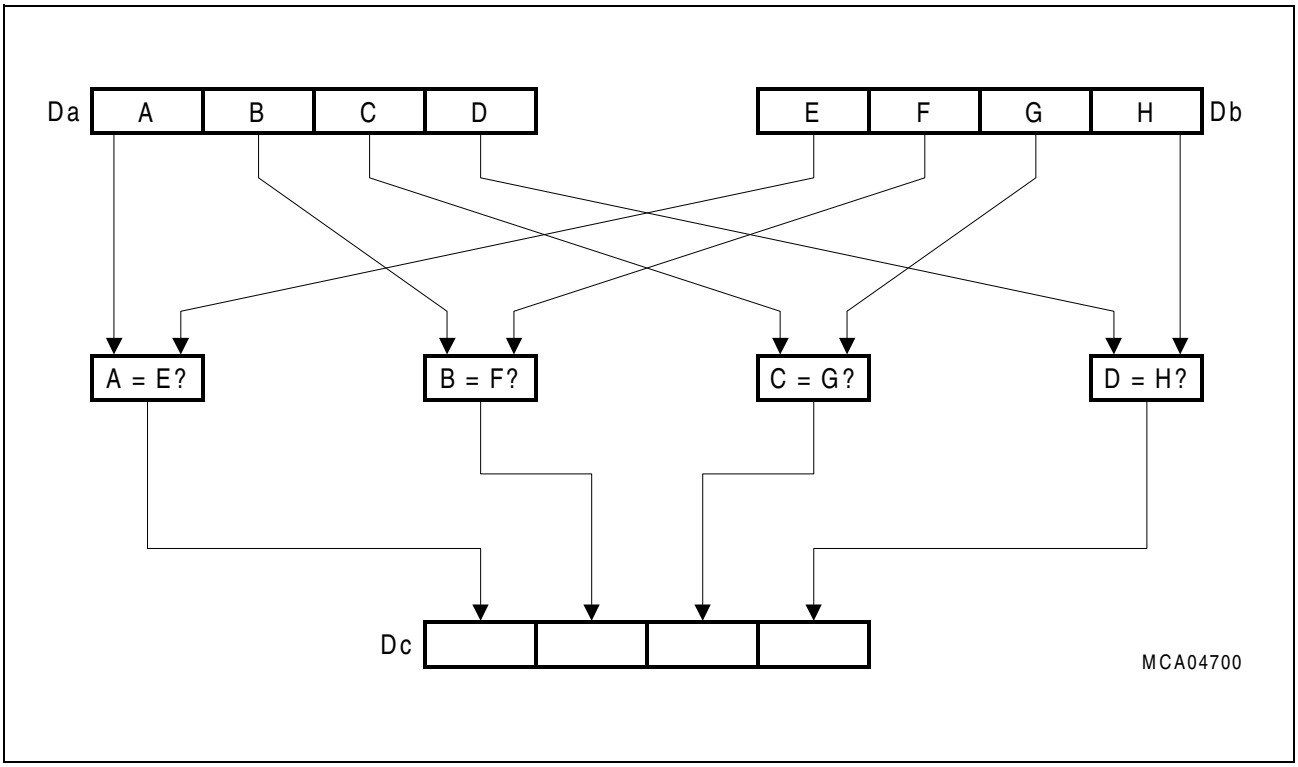


Figure 2-18 EQ.B Instruction Operation

2.3.3 Bit Operations

Instructions are provided that operate on single bits, denoted in the instruction mnemonic by the “T” data type modifier (for example, AND.T). There are eight instructions for combinatorial logic functions with two inputs, eight instructions with three inputs, and eight with two inputs and a shift. The one-bit result of a two-input function (for example, AND.T) is stored in the least significant bit of the destination data register, and the most-significant 31 bits are set to zero. The source bits can be any bit of any data register. This is illustrated in **Figure 2-19**. The available Boolean operations are: AND, NAND, OR, NOR, XOR, XNOR, ANDN, and ORN.

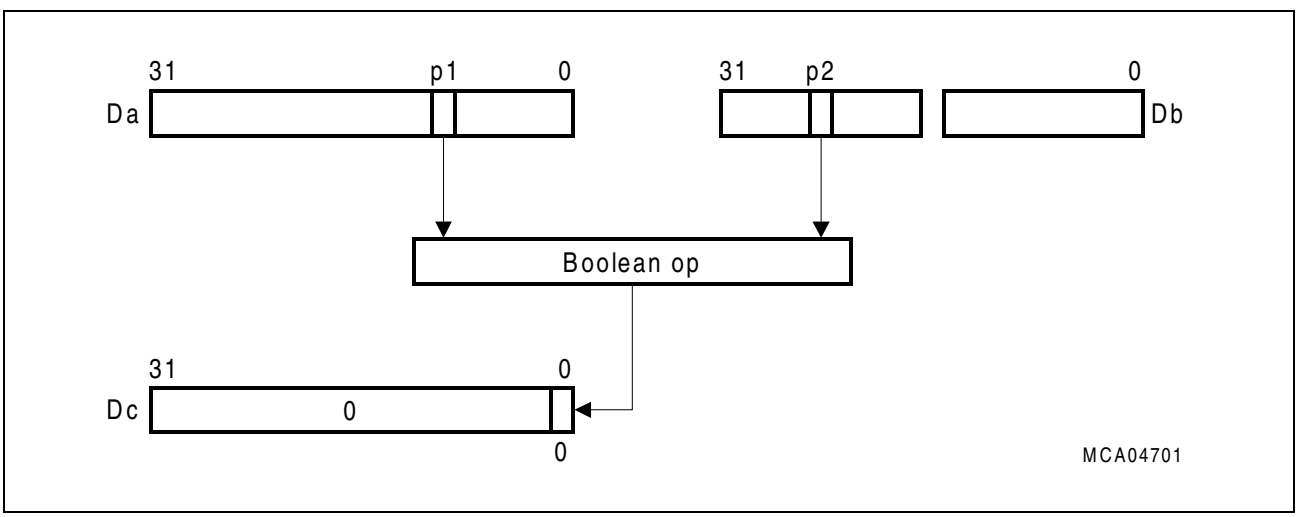


Figure 2-19 Boolean Operations

Evaluation of complex Boolean equations can use the 3-input Boolean operations in which the output of a two-input instruction is combined with the least significant bit of a third data register to form the input to a further operation. The result is written to bit 0 of the third data register, with the remaining bits unchanged (**Figure 2-20**).

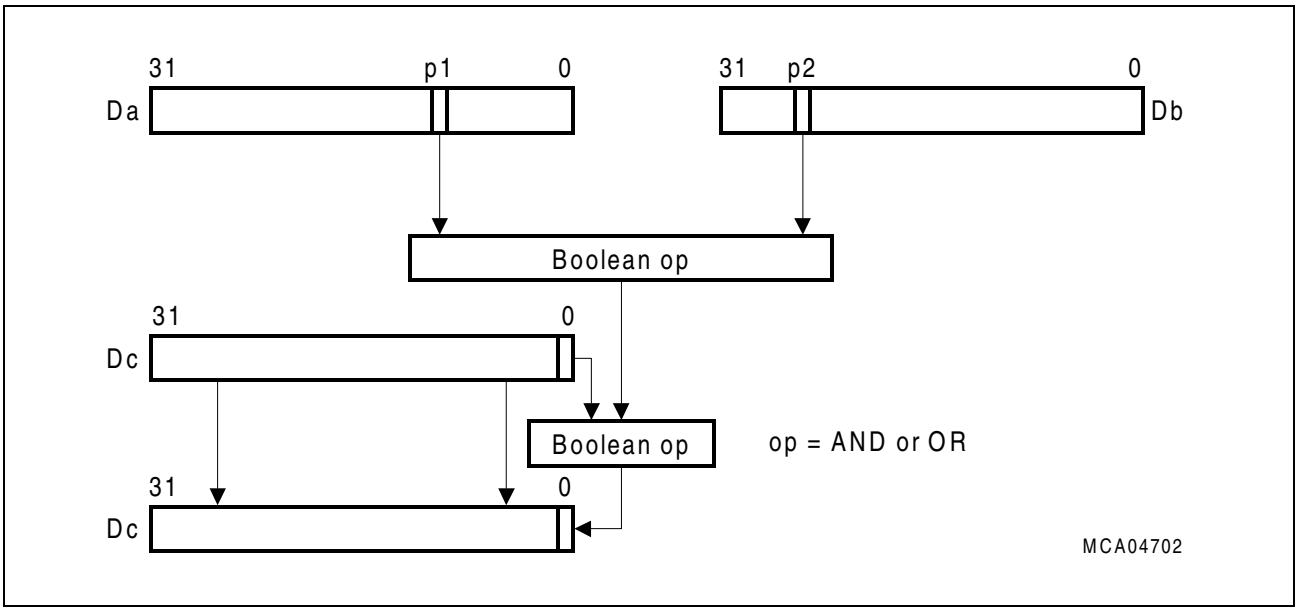


Figure 2-20 Three-Input Boolean Operation

Of the many possible three-input operations, eight have been singled out for the efficient evaluation of logical expressions. The eight instructions provided are: AND.AND.T, AND.ANDN.T, AND.NOR.T, AND.OR.T, OR.AND.T, OR.ANDN.T, OR.NOR.T, and OR.OR.T.

Just as for the comparison instructions, the results of bit operations often need to be packed into a single register for controller applications. For this reason, the basic two-input instructions can be combined with a shift prefix (for example, SH.AND.T). These operations first perform a single-bit left shift on the destination register and then store the result of the two-input logic function into its least significant bit (**Figure 2-21**).

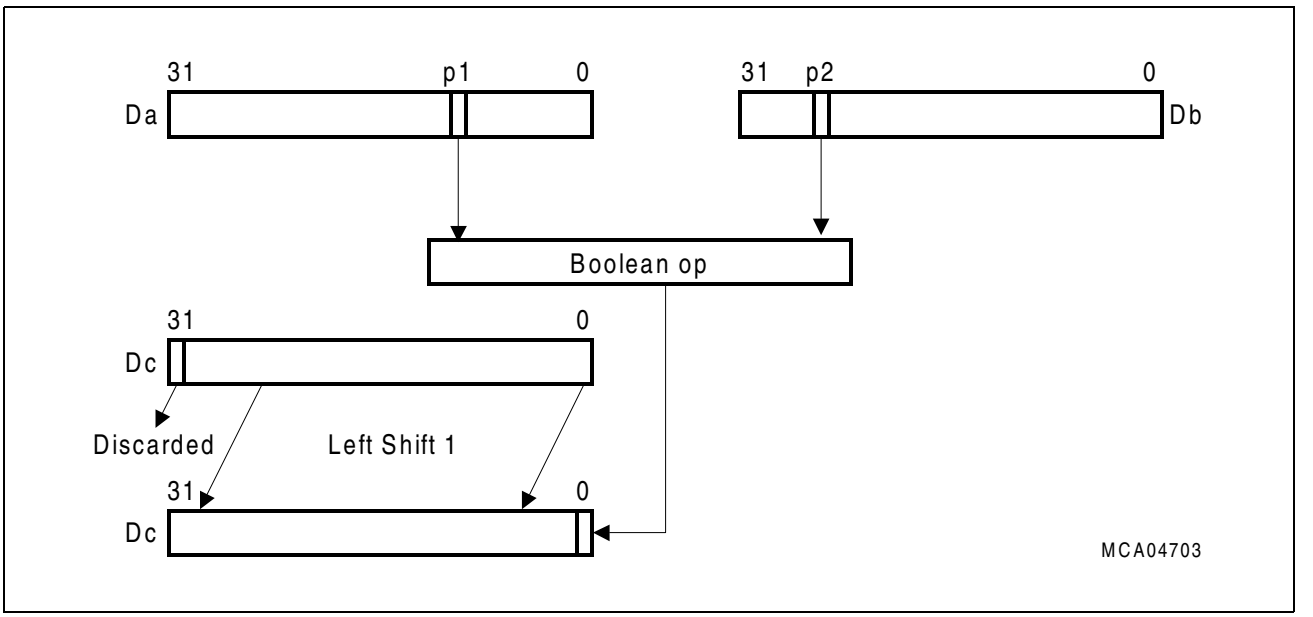


Figure 2-21 Shift Plus Boolean Operation

2.3.4 Address Arithmetic

The TriCore architecture provides selected arithmetic operations on the address registers. These operations supplement the address calculations inherent in the addressing modes used by the load and store instructions.

Initialization of base pointers requires loading a constant into an address register. When the base pointer is in the first 16 KBytes of each segment, this can be done using the Load Effective Address (LEA) instruction, using the absolute addressing mode. Loading a 32-bit constant into an address register can be accomplished using MOVH.A followed by an LEA that uses the base plus 16-bit offset addressing mode. For example,

```
movh.a   a5, ((ADDRESS+0x8000)>>16) & 0xffff
lea      a5, [a5](ADDRESS & 0xffff)
```

The MOVH.A instruction loads a 16-bit immediate into the most-significant 16-bits of an address register and zero-fills the least significant 16-bits. Adding a 16-bit constant to an address register can be done using the LEA instruction with the base plus offset addressing mode. Adding a 32-bit constant to an address register can be done in two instructions: an Add Immediate High Word (ADDIH.A), which adds a 16-bit immediate to the most-significant 16 bits of an address register, followed by an LEA using the base plus offset addressing mode. For example,

```
addih.a  a8, ((OFFSET+0x8000)>>16) & 0xffff
lea      a8, [a8](OFFSET & 0xffff)
```

The Add Scaled (ADDSC.A) instruction directly supports the use of a data variable as an index into an array of bytes, half-words, words, or double-words.

2.3.5 Address Comparison

As with the comparison instructions that use the data registers (see **Section 2.3.2**), the comparison instructions using the address registers put the result of the comparison in the least significant bit of the destination data register and clear the remaining register bits to zeros. An example of the Less Than (LT.A) instruction is shown in **Figure 2-22**.

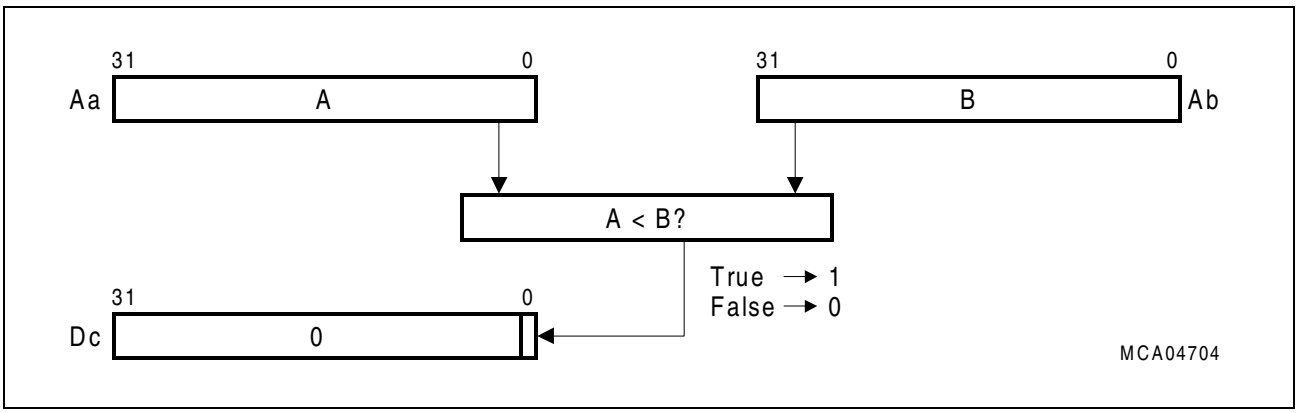


Figure 2-22 LT.A Comparison Operation

There are comparison instructions for equal (EQ.A), not equal (NE.A), less than (LT.A), and greater than or equal to (GE.A). As with the comparison instructions using the data registers, comparison conditions not explicitly provided in the instruction set can be obtained by swapping the two operand registers (**Table 2-5**).

Table 2-5 Comparison Operations

"Missing" Comparison Operation	TriCore Equivalent Comparison Operation
LE.A Dc, Aa, Ab	GE.A Dc, Ab, Aa
GT.A Dc, Aa, Ab	LT.A Dc, Ab, Aa

In addition to these instructions, instructions that test whether an address register is equal to zero (EQZ.A), or not equal to zero (NEZ.A) are supported. These instructions are useful to test for null pointers — a frequent operation when dealing with linked lists and complex data structures.

2.3.6 Branch Instructions

Branch instructions change the flow of program control by modifying the value in the PC register. There are two types of branch instructions: conditional and unconditional. Whether or not a conditional branch is taken depends on the result of a Boolean compare operation (see **Section 2.3.2**) rather than on the state of condition codes.

2.3.6.1 Unconditional Branch

There are three groups of unconditional branch instructions: Jump instructions, Jump and Link instructions, and Call and Return instructions.

A Jump instruction simply loads the Program Counter with the address specified in the instruction. A Jump and Link instruction does the same, and also stores the address of the next instruction in the “return address register” A11/RA. A Jump and Link instruction can be used to implement a subroutine call when the called routine does not modify any of the caller’s non-volatile registers. The Call instructions differ from a Jump and Link in that they save the caller’s non-volatile registers in a dynamically-allocated save area. The Return instruction, in addition to performing the return jump, restores the non-volatile registers.

Each group of unconditional Jump instructions contains separate instructions that differ in how the target address is specified. There are instructions using a relative 24-bit signed displacement (J, JL, and CALL), instructions using 24 bits of displacement as an absolute address (JA, JLA, and CALLA), and instructions using the address contained in an address register (JI, JLI, CALLI, RET, and RFE).

There are additional 16-bit instructions for a relative jump using an 8-bit displacement (J), an instruction for an indirect jump (JI), and an instruction for a return (RET).

Both the 24-bit and 8-bit relative displacements are scaled by two before they are used, because all instructions must be aligned on an even address. The use of a 24-bit displacement is shown in **Figure 2-23**.

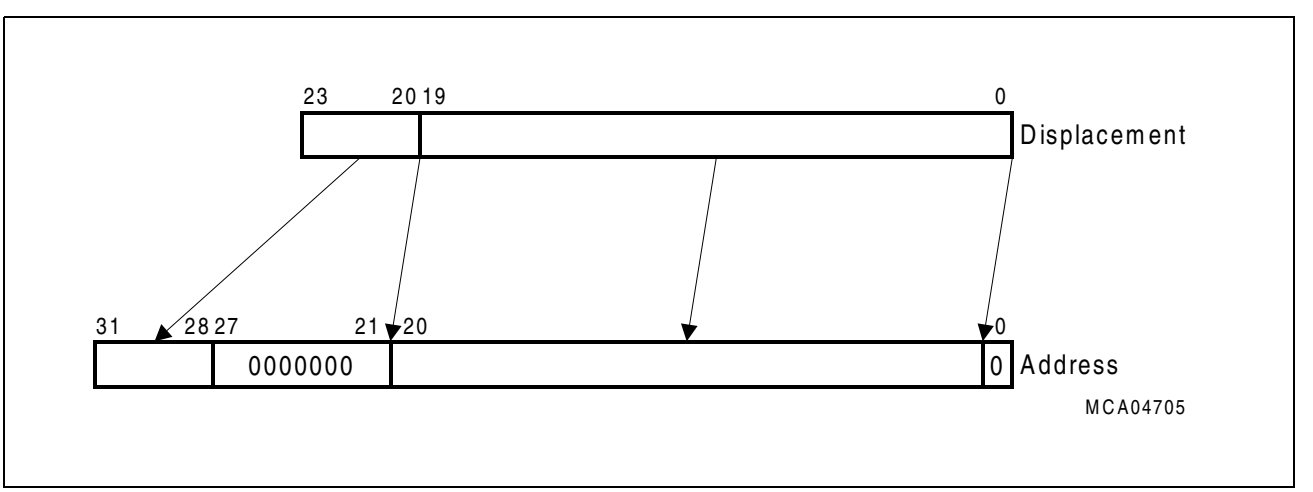


Figure 2-23 Displacement as Absolute Address

2.3.6.2 Conditional Branch

The conditional branch instructions use the relative addressing mode, with a displacement value encoded in 4, 8, or 15 bits. The displacement is scaled by 2 before it is used, because all instructions must be aligned on an even (half-word) address. The scaled displacement is sign-extended to 32 bits before it is added to the program counter, unless otherwise noted.

The Boolean test uses the contents of data registers, address registers, or individual bits in data registers.

Conditional Jumps on Data Registers

Six of the Conditional Jump instructions use a 15-bit signed displacement field: comparison for equality (JEQ), non-equality (JNE), less than (JLT), less than unsigned (JLT.U), greater than or equal (JGE), and greater than or equal unsigned (JGE.U). The second operand to be compared may be an 8-bit sign- or zero-extended constant. There are two 16-bit instructions that test whether the implicit D15 register is equal to zero (JZ) or not equal to zero (JNZ). The displacement is 8-bit in this case. Another two 16-bit instructions compare the implicit D15 register with a 4-bit, sign-extended constant (JEQ, JNE). The jump displacement field is limited to 4 zero-extended bits in this case.

There is a full set of 16-bit instructions that compare a data register to zero: JZ, JNZ, JLTZ, JLEZ, JGTZ, and JGEZ. Because any data register may be specified, the jump displacement is limited to 4-bit zero-extended constant in this case.

Conditional Jumps on Address Registers

The Conditional Jump instructions that use address registers are a subset of the data register Conditional Jump instructions. Four Conditional Jump instructions use a 15-bit signed displacement field: comparison for equality (JEQ.A), non-equality (JNE.A), equal to zero (JZ.A), and non-equal to zero (JNZ.A).

Because testing pointers for equality to zero is so frequent, two 16-bit instructions are provided (JZ.A and JNZ.A) with a displacement field limited to four zero-extended bits.

Conditional Jumps on Bits

Conditional jumps can be performed based on the value of any bit in any data register. The JZ.T instruction jumps when the bit is clear, and the JNZ.T instruction jumps when the bit is set. For these instructions, the jump displacement field is 15 bits.

There are two 16-bit instructions that test any of the lower 16 bits in the implicit register D15 and have a displacement field of four zero-extended bits.

2.3.6.3 Loop Instructions

Four special versions of Conditional Jump instructions are intended for efficient implementation of loops. The JNEI and JNED instructions are like a normal JNE instruction, but with an additional increment or decrement operation of the first register operand. The increment or decrement operation is performed unconditionally after the comparison. The jump displacement field is 15 bits. For example, a loop that should be executed for D3 = 3, …, 10 can be implemented as follows:

```
  lea   d3,3
loop1:
  ...
  jnei  d3,10,loop1
```

The LOOP instruction is a special kind of jump that utilizes the special TriCore hardware that implements “zero overhead” loops. The LOOP instruction only requires execution time in the pipeline the first and last time it is executed (for a given loop). For all other iterations of the loop, the LOOP instruction has zero execution time. For example, a loop that should be executed 100 times may be implemented as:

```
  mova   a2,99
loop2:
  ...
  loop   a2,loop2
```

The LOOP instruction above requires execution cycles the first and 100th time it is executed, but the other 98 executions require no cycles.

Note that the LOOP instruction differs from the other Conditional Jump instructions in that it uses an address register for the iteration count, rather than a data register. This allows it to be used in filter calculations in which a large number of data register reads and writes occur each cycle. Using an address register for the LOOP instruction reduces the need for an extra data register read port.

The LOOP instruction has a 32-bit version using a 15-bit displacement field (left-shifted by one bit and sign-extended), and a 16-bit version that uses a 4-bit displacement field. Unlike other 16-bit relative jumps, the 4-bit value is one-extended rather than zero-extended, because this instruction is specifically intended for loops.

An unconditional variant of the LOOP instruction is provided (LOOPU) which utilizes the zero overhead LOOP hardware. Such an instruction is used at the end of a while LOOP body to optimize the jump back to the start of the while construct.

2.3.7 Load and Store Instructions

The Load and Store instructions use seven addressing modes to move data between registers and memory (Table 2-6). The addressing mode determines the effective byte address for the Load or Store instruction and any update of the base pointer address register.

Table 2-6 Addressing Modes

Addressing Mode	Syntax	Effective Address	Instruction Format
Absolute	constant	{offset18[17:14], 14'bo, offset 18[13:0]}	ABS
Base + Short Offset	[An]offset	A[a]+sign_ext(offset10)	BO
Base + Long Offset	[An]offset	A[a]+sign_ext(offset16)	BOL
Pre-increment	[+An]offset	A[a]+sign_ext(offset10)	BO
Post-increment	[An+]offset	A[a]	BO
Circular	[An+c]offset	A[b]+A[b+1][15:0] (b is even)	BO
Bit-reverse	[An+r]	A[b]+A[b+1][15:0] (b is even)	BO

2.3.7.1 Load/Store Basic Data Types

The TriCore architecture defines loads and stores for the basic data types — corresponding to bytes, half-words, words and double-words — as well as for signed fractions and addresses. The movement of data between registers and memory for the basic data types is illustrated in Figure 2-24. Note that when the data loaded from memory is smaller than the destination register (that is, 8- and 16-bit quantities), the data is loaded into the least significant bits of the register (except for fractions which are loaded into the most significant bits of a register), and the remaining register bits are sign- or zero-extended to 32 bits, depending on the particular instruction.

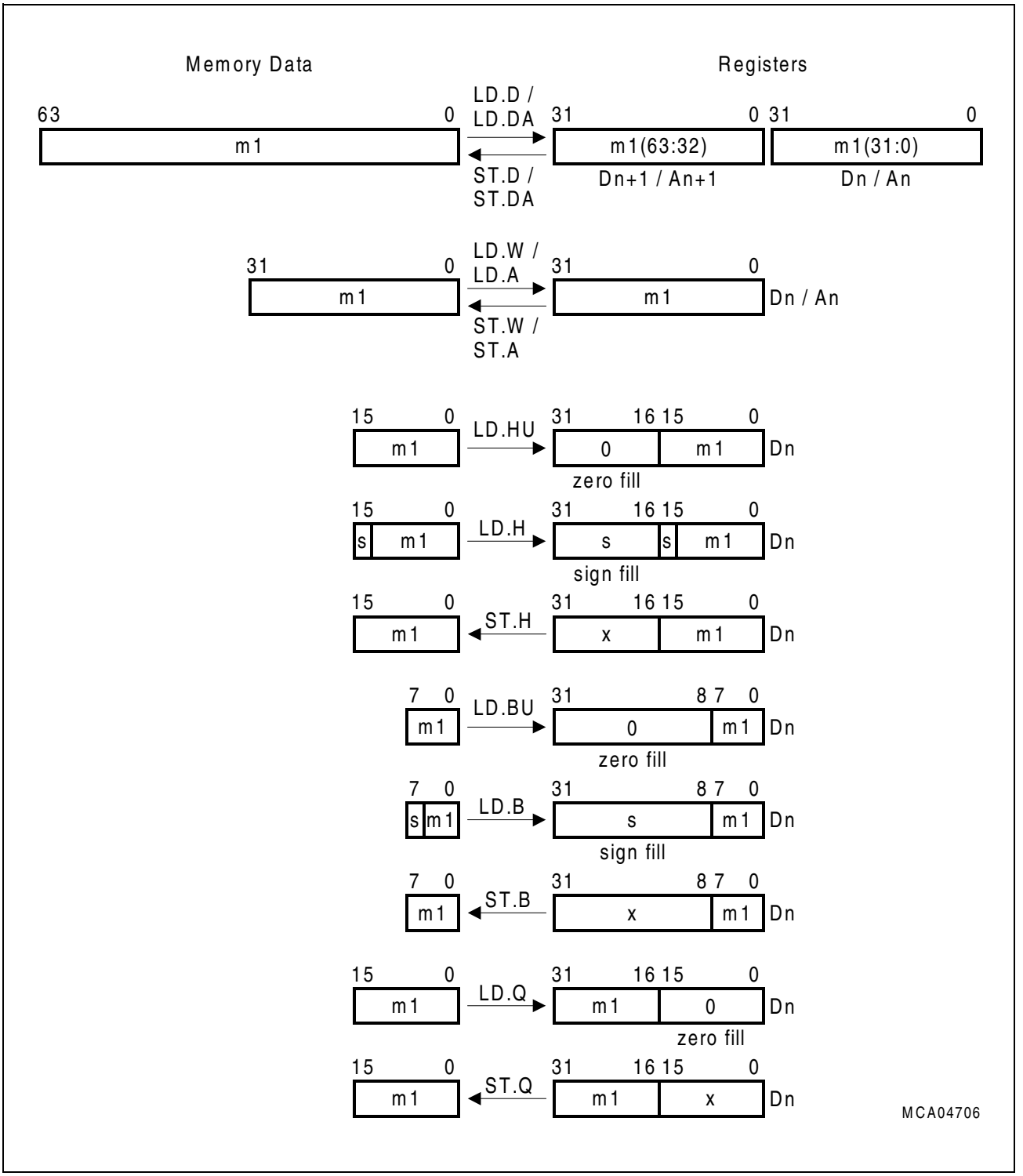


Figure 2-24 Load/Store Basic Data Types

2.3.7.2 Load Bit

The approach used to load individual bits depends on whether the bit within the word (or byte) is given statically or dynamically.

Loading a single bit with a fixed bit offset from a byte pointer is accomplished with an ordinary load instruction. One then can extract, logically operate on, or jump on any bit in a register.

Loading a single bit with a variable bit offset from a word-aligned byte pointer is done with a special scaled offset instruction. This offset instruction shifts the bit offset to the right by three positions (producing a byte offset), adds this result to the byte pointer above, and finally zeroes out the two lower bits, thus, aligning the access on a word boundary. A word load can then access the word that contains the bit which can be extracted with an extract instruction. The extract instruction uses only the lower five bits of the bit pointer, that is, the bits that were either shifted out or masked out above. An example is:

```
ADDSC.AT    A8,A9,D8       ; A9 = byte pointer. D8 = bit offset.
LD.W        D9,[A8]
EXTR.U      D10,D9,D8,1    ; D10[0] = loaded bit.
```

2.3.7.3 Store Bit and Bit Field

The ST.T instruction can clear or set single memory or peripheral bits, resulting in reduced code size. ST.T statically specifies a byte address and a bit number within that byte, and indicates whether the bit should be set or cleared. The addressable range for this instruction is the first 16 KBytes of each of the 16 memory segments.

Using any of the addressing modes, the Insert Mask (IMASK) instruction can be used in conjunction with the Load-Modify-Store (LDMST instruction) to store a single bit or a bit field to a location in memory. This operation is especially useful for reading and writing memory-mapped peripherals. The IMASK instruction is very similar to the INSERT instruction, but IMASK generates a data register pair that contains a mask and a value. The LDMST instruction uses the mask to indicate which portion of the word to modify. An example of a typical instruction sequence is:

```
imask    E8,3,4,2       ; insert value = 3, position = 4, width = 2
ldmst    _IOREG,E8      ; at absolute address "_IOREG"
```

To clarify the operation of the IMASK instruction, consider the following example. The binary value 1011_B is to be inserted starting at bit position 7 (the width is four). The IMASK instruction would result in the following two values:

```
0000 0000 0000 0000 0000 0111 1000 0000     MASK
0000 0000 0000 0000 0000 0101 1000 0000     VALUE
```

To store a single bit with a variable bit offset from a word-aligned byte pointer, first the word address is determined in the same way as for the load above. Next the special scaled offset instruction shifts the bit offset to the right by three positions — which

produces a byte offset — then adds this offset to the byte pointer above, and finally zeroes out the two lower bits, thus aligning the access on a word boundary. An IMASK and LDMST instruction can store the bit into the proper position in the word. An example is:

```
ADDSC.AT    A8,A9,D8        ; A9 = byte pointer. D8 = bit offset.
IMASK       E10,D9,D8,1     ; D9[0] = data bit.
LDMST       [A8],E10
```

2.3.8 Context Related Instructions

As well as the instructions that implicitly save and restore contexts (such as Calls and Returns), the TriCore instruction set includes instructions that allow a task's contexts to be explicitly saved, restored, loaded, and stored. These instructions are detailed in the following sections.

2.3.8.1 Context Saving and Restoring

The Upper Context of a task is always automatically saved on a call, interrupt, or trap. It is automatically restored on a return. However, the Lower Context of a task must be saved/restored explicitly.

The SVLCX instruction (Save Lower Context) saves registers A2 through A7 and D0 through D7 together with the return address in register A11/RA and the PCXI. This operation is performed when using the FCX and PCX pointers to manage the CSA lists.

The RSLCX instruction (Restore Lower Context) restores the Lower Context. It loads registers A2 through A7 and D0 through D7 from the CSA. It also loads A11/RA from the saved PC field. This operation is performed when using the FCX and PCX pointers to manage the CSA lists.

The BISR instruction (Begin Interrupt Service Routine) enables the interrupt system (ICR.IE is set to one), allows the modification of the CPU priority number (CCPN), and saves the Lower Context in the same manner as the SVLCX instruction.

2.3.8.2 Context Loading and Storing

The effective address of the memory area in which the context is stored to or loaded from is part of the Load or Store instruction. The effective address must resolve to a memory location aligned on a 16-word boundary; otherwise a data address alignment trap (ALN) is generated.

The STUCX instruction (Store Upper Context) stores the same context information that is saved with an implicit Upper Context save operation: Registers A10 – A15 and D8 – D15, and the current PSW and PCXI.

The LDUCX instruction (Load Upper Context) loads registers A10 – A15 and D8 – D15. The PSW and link word fields in the saved context in memory are ignored. The PSW, FCX, and PCXI are unaffected.

The STLCX instruction (Store Lower Context) stores the same context information that is saved with an explicit Lower Context save operation: Registers A2 – A7 and D0 – D7, together with the return address (RA) in A11 and the PCXI. The LDLCX instruction (Load Lower Context) loads registers A2 through A7 and D0 through D7. The saved return address and the link word fields in the context stored in memory are ignored. Registers A11/RA, FCX, and PCXI are not affected.

2.3.9 System Instructions

The system instructions allow user-mode and supervisor-mode programs to access and control various system services, including interrupts, and the TriCore's debugging facilities. There are also instructions that read and write the core registers, for both user and supervisor-only mode programs.

2.3.9.1 System Call

The SYSCALL instruction generates a system call trap, providing a secure mechanism for user-mode application code to request supervisor services. The system call trap — like other traps — vectors to the trap handler table, using the three-bit hardware-furnished trap class ID as an index. The trap class ID for system call traps is six. The trap identification number (TIN) is specified by an immediate constant in the SYSCALL instruction, and serves to identify the specific supervisor service that is being requested.

2.3.9.2 Synchronization Primitives

The TriCore architecture provides two synchronization primitives. These primitives provide a mechanism to software through which it can guarantee the ordering of various events within the machine.

DSYNC

The first primitive, DSYNC, provides a mechanism through which a data memory barrier can be implemented. The DSYNC instruction guarantees that all data accesses associated with instructions semantically prior to the DSYNC instruction are completed before any data memory accesses associated with an instruction semantically after DSYNC are initiated. This includes all accesses to the system bus and local data memory.

ISYNC

The second primitive, ISYNC, provides a mechanism through which the following can be guaranteed:

- If an instruction semantically prior to ISYNC make a software visible change to a piece of architectural state, then the effects of this change are seen by all instructions semantically after ISYNC. For example, if an instruction changes a code range in the

protection table, the use of an ISYNC will guarantee that all instructions after the ISYNC are fetched and matched against the new protection table entry.
- All cached states in the pipeline, such as loop cache buffers, are invalidated.

Operation of the ISYNC instruction is thus described as follows:

1. Wait until all instructions semantically prior to the ISYNC have completed.
2. Flush the CPU pipeline and cancel all instructions semantically after the ISYNC.
3. Invalidate all cached state in the pipeline.
4. Prefetch the next instruction after the ISYNC.

2.3.9.3 Access to the Core Special Function Registers

The TriCore accesses the CSFRs through two instructions: MFCR and MTCR. The MFCR instruction (Move From Core Register) moves the contents of the addressed CSFR into a data register. MFCR can be executed at any privilege level. The MTCR instruction (Move To Core Register) moves the contents of a data register to the addressed CSFR. To prevent unauthorized writes to the CSFRs, the MTCR instruction can only be executed at the supervisor privilege level.

The CSFRs are also mapped into the top of segment 15 in the memory address space. This mapping makes the complete architectural state of the core visible in the address map, which allows efficient debug and emulator support.

Note: It is not permitted for the core to access the CSFRs through this mechanism; it must use MFCR and MTCR.

There are no instructions allowing bit, bit field, or load-modify store accesses to the CSFRs. The RSTV instruction (Reset Overflow Flags) resets the overflow flags in the PSW, without modifying any of the other bits in the PSW. This instruction can be executed at any privilege level.

2.3.9.4 Enabling/Disabling the Interrupt System

For non-interruptible operations, the ENABLE and DISABLE instructions allow the explicit enabling and disabling of interrupts in user and supervisor modes. While disabled, an interrupt will not be taken by the CPU regardless of the relative priorities of the CPU and the highest interrupt pending. The only “interrupt” that will be serviced while interrupts are disabled is the NMI (non-maskable interrupt) since it is a trap.

If a user process accidentally disables interrupts for longer than a specified time, the Watchdog Timer can be used to recover.

Programs executing in supervisor mode can use the 16-bit Begin ISR (BISR) instruction to save the Lower Context of the current task, set the current CPU priority number, and re-enable interrupts (which are disabled by the processor when an interrupt is taken).

2.3.9.5 RET and RFE

The function return instruction (RET) is used to return from a function that was invoked via a CALL instruction. The return from exception instruction (RFE) is used to return from an interrupt or trap handler. The two instructions perform very similar operations; they restore the Upper Context of the calling function or interrupted task, and branch to the return address contained in register A11 (prior to the context restore operation). The instructions differ in the error checking they perform for call depth management. Issuing an RFE instruction when the current call depth (as tracked in the PSW) is nonzero generates a context nesting error trap. Conversely, a context call depth underflow trap is generated when an RET instruction is issued when the current call depth is zero.

2.3.9.6 Trap Instructions

The Trap on Overflow (TRAPV) and Trap on Sticky Overflow (TRAPSV) instructions can be used to cause a trap if the PSW's V and SV bits, respectively, are set (**Section 2.3.1**).

2.3.9.7 No Operation

Although there are many ways to represent a no-operation (for example, adding zero to a register), an explicit NOP instruction is included so that it can be easily recognized, allowing the CPU to minimize power consumption during its execution. For example, a sequence of NOP instructions in a loop could be used as a low-power state that has a very fast interrupt response time.

2.3.10 16-Bit Instructions

The 16-bit instructions are a subset of the 32-bit instruction set, chosen because of their frequency of static use. They significantly reduce static code size and thus provide a reduction in the cost of code memory and a higher effective instruction bandwidth. Because the 16-bit and 32-bit instructions all differ in the primary opcode, the two instruction sizes can be freely intermixed.

The 16-bit instructions are formed by imposing one or more of the following format constraints: smaller constants, smaller displacements, smaller offsets, implicit source, destination, or base address registers, and combined source and destination registers (the 2-operand format). In addition, the 16-bit load and store instructions support only a limited set of addressing modes.

The registers D15 and A15 are used as implicit registers in many 16-bit instructions. For example, there is a 16-bit compare instruction (EQ) that puts a Boolean result in D15, and a 16-bit conditional move instruction (CMOV) which is controlled by the Boolean in D15.

The 16-bit load and store instructions are limited to the register indirect (base plus zero offset), base plus offset (with implicit base or source/destination register), and post-increment (with default offset) addressing modes.

2.4 CPU Pipelines

This section describes the TC1765 CPU pipelines including the integer and load/store pipelines, and the loop pipeline.

2.4.1 CPU Pipeline Overview

As specified by the TriCore architecture, the TC1765 implements a pipelined, superscalar processor architecture that allows the execution of up to three instructions in parallel. The processor pipeline design reduces branch latency, data dependencies, and overall system complexity.

Two major pipelines perform integer operations and load/store operations. Each of these has four stages: Fetch (common to both), Decode, Execute, and Write-back. A third minor pipeline optimizes DSP loops. The three pipelines are illustrated in **Figure 2-25**.

2.4.2 Integer and Load/Store Pipelines

The Integer Pipeline executes the following operation types.

- Integer arithmetic and logical operations
- Bit-wise logical operations
- Multiply-accumulate (MAC) operations
- Integer division
- Conditional data jumps

The Load/Store Pipeline executes the following operation types.

- Load and Store operations
- Context-switch operations
- System operations
- Address arithmetic calculations
- Unconditional and conditional branch target calculations

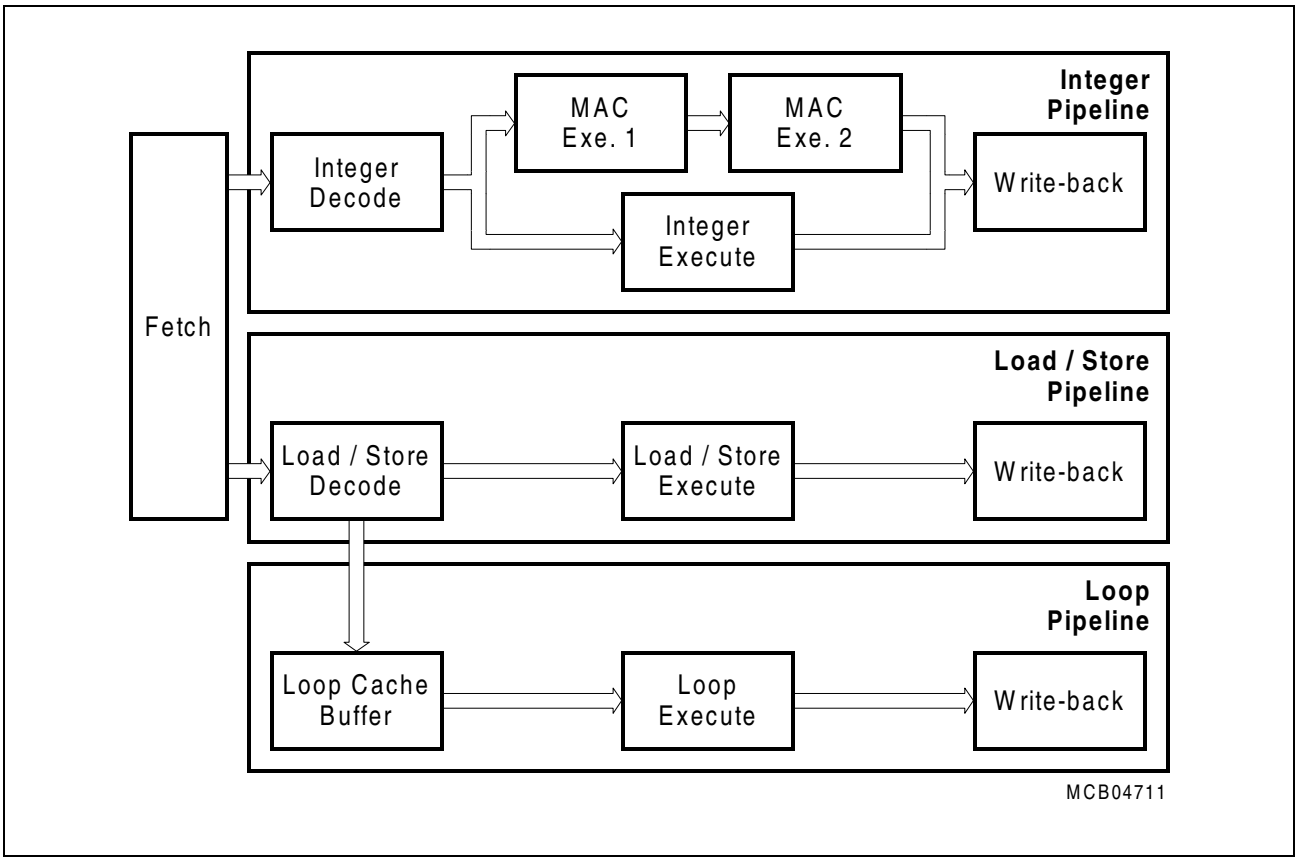


Figure 2-25 Pipeline Architecture

The pipelines share a common fetch stage that can issue one instruction to each pipeline per cycle. Certain issue constraints apply. For instance, when two instructions are issued in parallel, the first instruction must be an integer pipeline instruction. An integer ADD followed by a load instruction can be issued in parallel, but a load followed in the pair by an integer ADD cannot.

For example, the following code sequence takes four cycles.

```
add     d0, d1, d2
sub     d0, d0, d3
ld.w    d1, [a0]0
xor     d2, d1, d0
st.w    [a1]0, d2
ld.a    a0, [a5]4
```

Cycle	Integer	Load/Store
1	add	–
2	sub	ld.w
3	xor	st.w
4	–	ld.a

Note that on the third cycle, the XOR instruction and dependent store are dual-issued. The result from the XOR can be forwarded to the store instruction without any stall penalty. In general, all required forwarding paths are implemented so that dependent instructions can be executed without stall penalties.

All simple integer operations, bit operations, and address arithmetic instructions execute in a single cycle. Divide instructions, such as DVSTEP, require eight uninterruptable cycles to execute.

The multiply-accumulate (MAC) instructions are executed in a special two-stage MAC pipeline. The first stage contains two 16 × 16-bit multipliers. The second stage contains the accumulation, rounding and saturation logic. The MAC pipeline can perform a 32 × 32-bit multiply every two cycles with a latency of three cycles, or two 16 × 16-bit multiplies every cycle with a latency of two cycles.

2.4.3 Loop Pipeline

The Loop Pipeline optimizes the execution of loops, such as those typically found in DSP applications. This pipeline is driven by the Loop Cache Buffer (LCB), which stores the location, target, and other required information. The loop instruction is executed in the Load/Store Pipeline on its first iteration, and in the loop pipeline thereafter. If the loop is single-issued, the LCB is updated when it is detected in the decode stage of the pipeline. On subsequent iterations of the loop, when the LCB detects the end of the loop, it automatically fetches the start of the loop body. Unlike a normal Branch Target Buffer hit, the loop instruction itself is not fetched. It is injected from the LCB into the Loop Pipeline during the last execute cycle.

For example, the following code will execute as shown below:

```
        mov.a   a0, number_of_iterations - 1
loop_start:
        add     d0, d0, d1
        ld.w    d1, [a0+]4
        loop    d0, loop_start
```

Cycle	Integer	Load/Store	Loop
1	–	mov.a	–
2	add	ld.w	–
3	–	loop	–
4	–	–	–
5	add	ld.w	loop
6	add	ld.w	loop
7	add	ld.w	loop
8	add	ld.w	loop

As can be seen, after the first pass through the loop, each subsequent iteration will take only one clock cycle, thereby providing zero overhead loop capability.

2.4.4 Context Operations

Context save and context restore operations associated with calls, returns, interrupts, and so on, use the 128-bit data bus between the register file and the local on-chip data memory. The CPU contains dedicated hardware to optimize context switching, resulting in a context-save time to the on-chip local memory of between two and four cycles.

3 Clock System

This chapter describes the TC1765's clock system. Topics covered include clock gating, clock domains, clock generation, the operation of clock circuitry, boot-time operation, fail-safe operation, clock control registers, and power management.

The TC1765 clock system performs the following functions:

- Acquires and buffers incoming clock signals to create a master clock frequency
- Distributes in-phase synchronized clock signals throughout the TC1765's entire clock tree
- Divides a system master clock frequency into lower frequencies required by the different modules for operation
- Dynamically reduces power consumption during operation in some functional units
- Statically reduces power consumption through programmable power-saving modes
- Reduces electromagnetic interference (EMI)

The clock system must be operational before the TC1765 can function, so it contains special logic to handle power-up and reset operations. Its services are fundamental to the operation of the entire system, so it contains special fail-safe logic.

Figure 3-1 shows the structure of the TC1765 clock system. The system clock f_{SYS} is generated by the oscillator circuit and the phase-locked loop (PLL) unit. The module clocks are all derived from the system clock. Each peripheral module can define a specific operation frequency of its module clock f_{MOD}.

The functionality of the control blocks shown in **Figure 3-1** varies depending on the functional unit being controlled. Some functional units, such as the FPI Bus or the watchdog timer, are directly driven by the system clock Detailed descriptions on the clock control register options for each unit are described in **Section 3.2**.

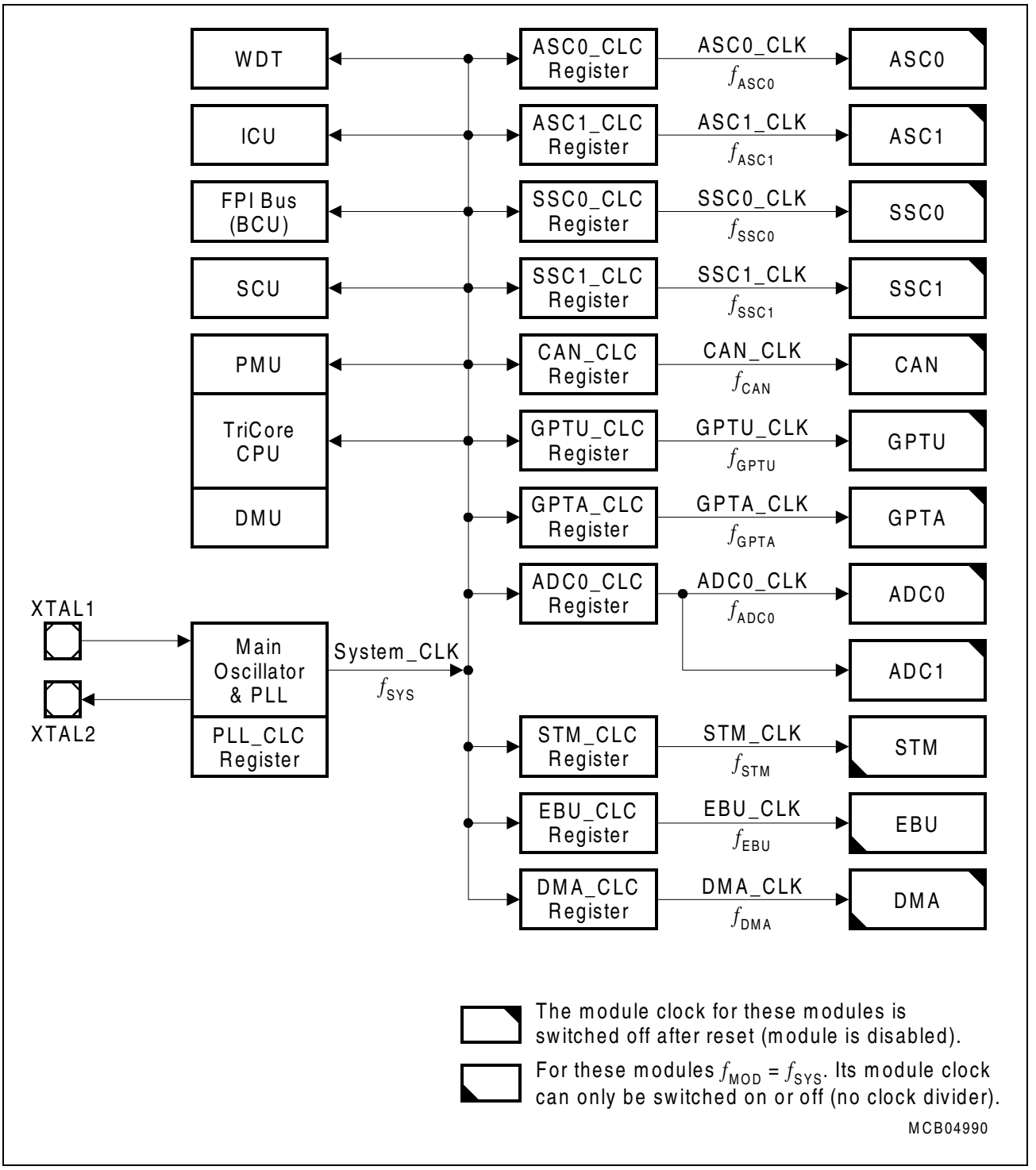


Figure 3-1 TC1765 Clocking System

3.1 Clock Generation Unit

The Clock Generation Unit in the TC1765, shown in **Figure 3-2**, consists of an oscillator circuit and a Phase-Locked Loop (PLL). The PLL can convert a low-frequency external clock signal to a high-speed internal clock for maximum performance. The PLL also has fail-safe logic that detects degenerate external clock behavior such as abnormal frequency deviations or a total loss of the external clock. It can execute emergency actions if it looses its lock on the external clock.

In general, the clock generation unit is controlled through the System Control Unit (SCU) module of the TC1765.

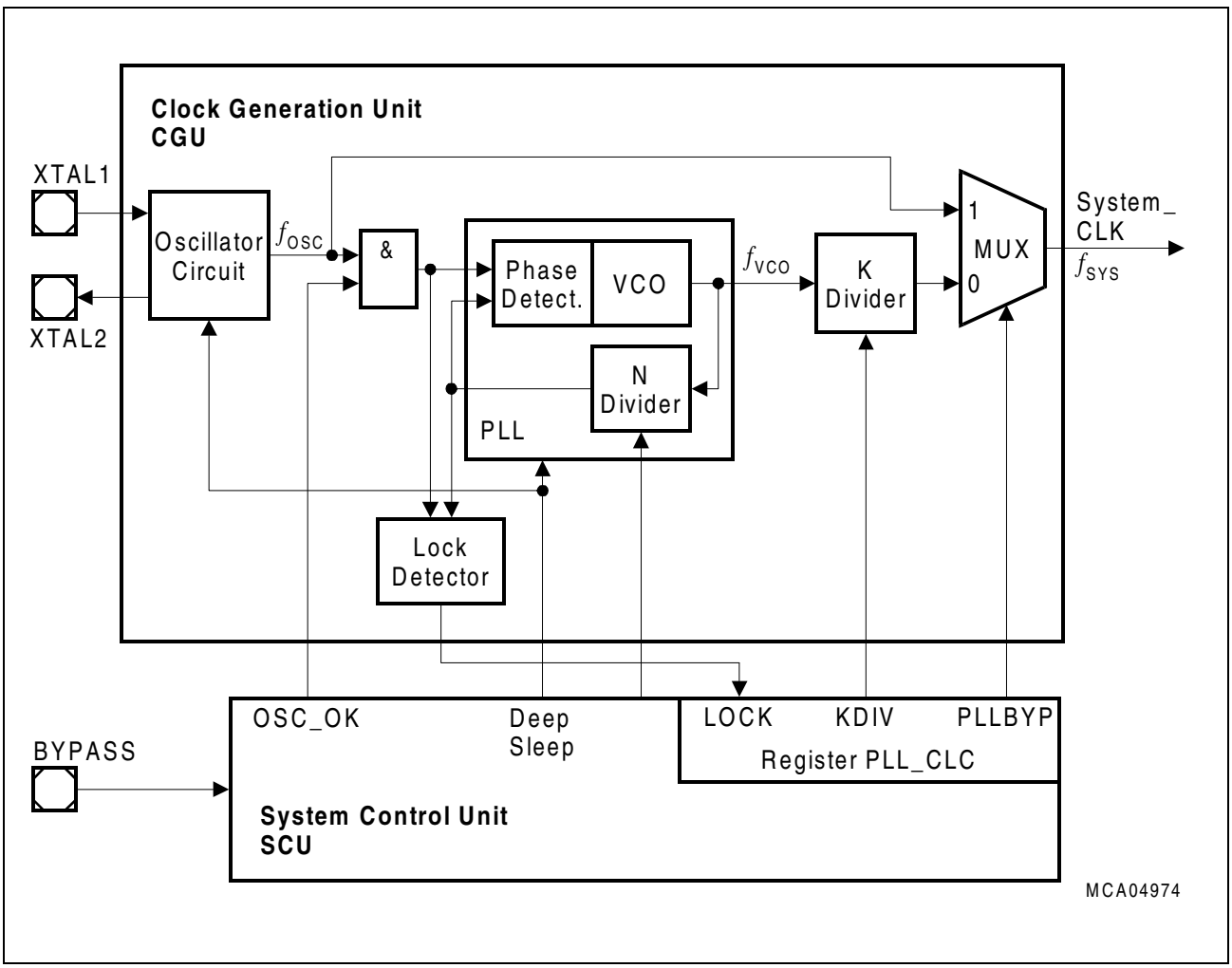


Figure 3-2 Clock Generation Unit Block Diagram

The following sections give descriptions of the various blocks of the clock generation unit.

3.1.1 Oscillator Circuit

The oscillator circuit, designed to work with both, an external crystal oscillator or an external stable clock source, basically consists of an inverting amplifier with XTAL1 as input, and XTAL2 as output.

When using a crystal, a proper external oscillator circuitry must be used, connected to both pins, XTAL1 and XTAL2. The on-chip oscillator frequency can be within the range of 4 MHz to 16 MHz. When using an external clock signal it must be connected to XTAL1. XTAL2 is left open (unconnected). If the external clock signal at XTAL1 is stopped e.g. for test purposes (not a normal operating condition), XTAL1 should be held at high level. For direct drive operation without PLL, the frequency of an external clock must not exceed 40 MHz.

Further specifications on the frequency limits of the clock circuitry are given in the TC1765 device specifications (Data Sheet).

Figure 3-3 shows the recommended external oscillator circuitries for both operating modes, external crystal mode and external input clock mode.

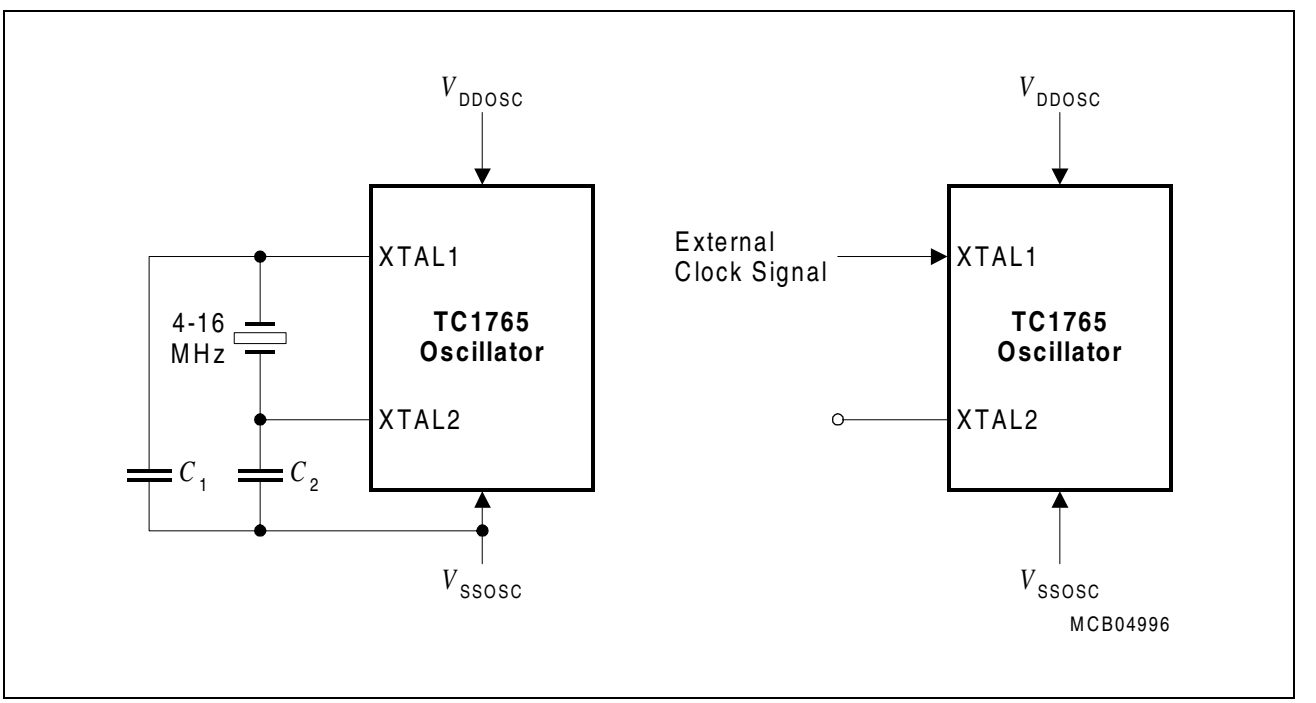


Figure 3-3 TC1765 Main Oscillator Circuitries

3.1.2 Phase-Locked Loop (PLL)

The PLL consists of a voltage controlled oscillator (VCO) with a feedback path. A divider in the feedback path divides the VCO frequency down. The resulting frequency is then compared to the externally applied frequency. The phase detection logic determines the difference between the two clock signals and accordingly controls the frequency of the VCO. During start-up, the VCO increases its frequency until the divided feedback clock matches the external clock frequency. A lock detection logic monitors and signals this condition. The phase detection logic continues to monitor the two clock signals and adjusts the VCO clock if required.

Due to this operation, the VCO clock of the PLL has a frequency which is a multiple of the externally applied clock. The factor for this is controlled through the value applied to the divider in the feedback path. That is why this factor is often called a multiplier, although it actually controls a divider.

3.1.2.1 N-Divider

The feedback divider has a fixed divider value of N = 10.

3.1.2.2 VCO Frequency Ranges

Stable and reliable operation of the VCO, and minimization of the jitter (the frequency variations of the VCO output between adjustment points), is critical for precise clock generation. To provide optimum behavior, the following frequency range for f_{VCO} must be selected:

$$150\ \text{MHz} \leq f_{VCO} \leq 200\ \text{MHz} \qquad [3.1]$$

3.1.2.3 Lock Detection

A lock detector circuit determines whether the PLL is locked appropriately to the external clock signal, and indicates the PLL lock state to the SCU. If the PLL looses synchronization to the external clock due to a failure of the external clock, the SCU detects this case and shuts off the oscillator input to the VCO via deactivation of the OSC_OK signal.

3.1.2.4 K-Divider

The K-Divider is a software controlled divider. The bit field KDIV is provided in register PLL_CLC. Software can write to this field in order to change the system frequency f_{SYS}. **Table 3-3** lists the possible values for KDIV and the resulting division factor.

The divider is designed such that a synchronous switching of the clock is performed without spurious or shortened clock pulses when software changes the divider factor KDIV. However, special attention has to be paid concerning the effect of such a clock

change to the various modules in the system. For instance, changing the clock frequency while an external memory access is performed by the EBU could result in a failure of the access. It is strongly recommended to perform clock frequency changes only when no critical system operations are in progress to avoid hazardous effects.

3.1.2.5 Clock Source Control

The clock system provides three ways for the generation of the system clock:

- **PLL Bypass Operation:**
  The system clock is directly derived from the oscillator clock. In this case $f_{SYS} = f_{OSC}$. This option is controlled through the signal PLL_BYPASS.
- **PLL Operation:**
  The system clock is derived from the oscillator clock, but multiplied by the PLL and optionally divided by the K-divide: $f_{SYS} = f_{OSC} \times N / KDIV$. PLL_BYPASS must be inactive for this PLL operation.

The external PLL configuration input pin BYPASS is provided to enable the bypass options. This pin is sampled during a power-on reset. Its state is latched in register PLL_CLC with the rising edge of $\overline{PORST}$. Signal PLL_BYPASS is generated using the value of the BYPASS pin (the N-Divider is not used in the bypass mode). **Table 3-1** shows how the clock source options are selected.

Table 3-1 Clock Source Selection

Pin BYPASS	Selected Operation
0	**PLL Operation** System clock is generated by the PLL.
1	**PLL Bypass Operation** System clock generated directly by external clock.

Note: Using the internal oscillator and bypassing the PLL might cause spikes during waking-up the CPU from idle mode, sleep mode or deep sleep mode. This is due to the fact, that the output of the internal oscillator is not filtered.

3.1.3 Enable/Disable Control

If bypass mode or Deep Sleep Mode is selected, the PLL is shut off by the SCU via the DEEP_SLEEP signal. In Deep Sleep Mode, also the main oscillator circuit is disabled.

3.1.4 Determining the System Clock Frequency

This section gives the formulas for the determination of the system clock frequency for the three different clock source options.

3.1.4.1 PLL Bypass Operation

In PLL bypass operation, the system clock has exactly the same frequency as the external clock source:

$$f_{SYS} = f_{VCO} \quad [3.2]$$

It is recommended to use this mode only when using an external stable clock source. When using a crystal oscillator in this mode, the system clock might not have a duty cycle of 50% due to asymmetric thresholds and voltages at the oscillator circuitry. It has to be assured, that the minimum clock phase produced does not violate the specifications. Usually, a slower frequency than the specified maximum speed for the chip needs to be used in such a case.

3.1.4.2 PLL Operation

In PLL operation, the system clock is derived from the VCO frequency f_{VCO} divided by the K-factor. f_{VCO} is generated from the external clock multiplied by 10.

The system clock frequency f_{SYS} can be made proportional to the ratio 10 / K, where bit field PLL_CLC.KDIV determines the clock scale factor K. The selectable clock scale factors are summarized in **Table 3-2**.

Table 3-2 PLL Scale Factors

KDIV[2:0]	Selected K Factor
000	2
001	16
010	4
011	5
100	6
101	8
110	9
111	10

The VCO output frequency is determined by:

$$f_{VCO} = 10 \times f_{OSC} \quad [3.3]$$

and the resulting system clock is determined by:

$$f_{SYS} = f_{VCO} / K = (10 / K) \times f_{OSC} \quad [3.4]$$

Since stable operation of the VCO is only guaranteed if f_{VCO} remains inside of the defined frequency range for the VCO (see **Equation [3.1]**), the external frequency f_{OSC} is also confined to certain ranges. **Table 3-3** lists these ranges.

Table 3-3 Input Frequencies and N Factor for f_{VCO}

N-Factor	f_{VCO} = 150 MHz	f_{VCO} = 160 MHz	f_{VCO} = 200 MHz
10	f_{OSC} = 15 MHz	f_{OSC} = 16 MHz	f_{OSC} = 20 MHz[1)]

1) This combination should not be used because the maximum oscillator frequency of 16 MHz is exceeded.

Table 3-4 Output Frequencies f_{SYS} Derived from Various Output Factors

K-Factor		f_{SYS}			Duty Cycle [%]	Jitter
Selected Factor	KDIV	f_{VCO} = 150 MHz	f_{VCO} = 160 MHz	f_{VCO} = 200 MHz		
2	000_B	75[1)]	80[1)]	100[1)]	50	linear depending on f_{VCO}; at f_{VCO} = 200 MHz ±200 ps at f_{VCO} = 150 MHz ±250 ps additional jitter for odd K factors t.b.d.
4	010_B	37.5	40	50[1)]	50	
5[2)]	011_B	30	32	40	40	
6	100_B	24.5	26.67	33.33	50	
8	101_B	18.75	20	25	50	
9[2)]	110_B	16.67	17.78	22.22	44	
10	111_B	15	16	20	50	
16	001_B	9.38	10	12.5	50	

1) These combinations cannot be used because the maximum system clock of 40 MHz is exceeded.

2) These odd K-Factors should not be used (not tested because of the unsymmetrical duty cycle).

Note: Further PLL characteristics are given in the TC1765 device specifications (Data Sheet).

3.1.5 PLL Clock Control and Status Register

The PLL Clock Control and Status Register PLL_CLC is located in the address range reserved for the System Control Unit (SCU). It holds the hardware configuration bits of the PLL, latched at the end of power-on reset, and provides the control for the K-Factor as well as the PLL Lock status bit.

Note that register PLL_CLC is specially protected. In order to write to PLL_CLC, the WDT_CON0.ENDINIT bit must be set to 0 through a password-protected access mechanism to register WDT_CON.

The indicator “U” in the reset value of PLL_CLC indicates that the reset values for these bits are user-defined through the value applied to the PLL configuration pins.

PLL_CLC
PLL Clock Control Register **Reset Value: 0007 UU00$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0													KDIV		
r													rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PLL BYP	0	1	0	0	0	1	LO CK	0							
rh	r	r	r	r	r	r	rh	r							

Field	Bits	Type	Description
LOCK	8	rh	**PLL Lock Status Flag** 0 PLL is not locked 1 PLL is locked
PLLBYP	15	rh	**PLL Bypass Status Flag** Indicates the state of the BYPASS control input line latched with the rising edge of $\overline{\text{PORST}}$.
KDIV	[18:16]	rw	**PLL K-Factor Selection** Definitions see **Table 3-2**
1	9, 13	r	**Reserved**; returns 1 if read; any write operation to these bits has no effect.
0	[7:0], [12:10], 14, [31:16]	r	**Reserved**; returns 0 if read; any write operation to these bits has no effect.

3.1.6 Startup Operation

When power is switched on to the TC1765, a low level has to be applied to the power-on reset pin, $\overline{\text{PORST}}$. The part is asynchronously held in reset and the state of the PLL bypass pin, BYPASS, are enabled to control the operation of the clock circuitry.

If pin BYPASS is at a high level, the PLL is disabled, and direct clock input is selected. The low level at pin $\overline{\text{PORST}}$ has to be held long enough to make sure that a stable clock is provided to the TC1765. In case of an external crystal oscillator, it can take several ms until the oscillator has started up and is stable. If the clock input is provided by another clock source with faster startup characteristics, the requirements for the $\overline{\text{PORST}}$ low level can be relaxed accordingly.

If BYPASS is at a low level during power-on reset, both, the oscillator and the PLL start to operate. The voltage controlled oscillator (VCO) of the PLL will start up very quickly and generate an internal clock with the PLL base frequency. Usually, the oscillator with its external crystal and supporting external circuitry requires a time in the range of some ms to startup. As soon as it provides a stable clock frequency, the PLL will lock to this frequency. This state is signalled by setting bit PLL_CLC.LOCK.

Two situations are possible when $\overline{\text{PORST}}$ becomes inactive (low-to-high transition):

1. PLL is not locked (PLL_CLC.LOCK = 0):
   The PLL provides an emergency clock at PLL base frequency. The system clock will be at this PLL base frequency as long as the LOCK bit is not set. In this case, a program should wait for bit LOCK to be set before it proceeds time critical initialization procedures and operations.
2. PLL is locked (PLL_CLC.LOCK = 1):
   The PLL is already at its nominal frequency.

With the low-to-high transition of the signal at $\overline{\text{PORST}}$, the state of the pins BYPASS, $\overline{\text{OCDSE}}$, $\overline{\text{BRKIN}}$ and CFG[2:0] are latched internally in the SCU. The state of BYPASS and PLL_LOCK is latched into register PLL_CLC, while the state of the debug pins $\overline{\text{OCDSE}}$ and $\overline{\text{BRKIN}}$, and the state of the configuration pins CFG[2:0] is latched in the reset status register RST_SR.

3.1.7 PLL Loss of Lock Operation

The PLL provides mechanisms to detect a failure of the external clock and to bring the TC1765 into a safe state in such a case. If the PLL loses the lock to the external clock, either due to a break of the crystal or an external line, it resets its lock line PLL_LOCK. The clock control circuitry then sets the PLL Loss of Clock NMI flag (PLLNMI) in register NMISR and activates a NMI trap request to the CPU. In addition, it disables the oscillator input clock f_{OSC} to the PLL to avoid unstable operation due to noise or sporadic clock pulses coming from the oscillator circuit and the PLL still trying to lock onto this invalid clocks. Without having an input clock, the PLL gradually slows down to its base

frequency and remains there. While this frequency is defined within a certain frequency range, emergency actions can be taken by the CPU since the TC1765 is still clocked.

The TC1765 remains in this state until the next power-on reset through pin $\overline{\text{PORST}}$, where then the PLL tries to restart and lock to the external clock again. No other reset cause can terminate this loss-of-clock state to avoid unstable operation due to the PLL trying to lock again.

Note that this fail safe mechanism is only provided if the PLL is enabled (BYPASS = 0), but not if direct clock input is selected (BYPASS = 1).

3.2 Power Management and Clock Gating

Because power dissipation is related to the frequency of gate transitions, the TC1765 performs power management principally by clock gating - that is, controlling whether the clock is supplied to its various functional units. Gating off the clock to unused functional modules also reduces electromagnetic interference (EMI) since EMI is related to both the frequency and the number of gate transitions.

Clock gating is done either dynamically or statically. Dynamic clock gating in this context means that the TC1765 itself enables or disables clock signals within some functional modules to conserve power. Static gating means that software must enable or disable clock signals to functional modules. Clock gating is performed differently at different levels of system scope: dynamic gating is generally performed at the lowest levels, either within a small region of logic, or at functional-unit boundaries for uncomplicated functions where hardware can dynamically determine whether that functionality is required, and can enable or disable it appropriately without software intervention. Static gating - which requires software intervention - is used to enable or disable clock delivery to individual high-level functional units, or to disable clock delivery globally at the clock's source. When the clock to individual functional units is gated off, they are said to be in Sleep Mode. When the TC1765's clock is gated off at its source, the TC1765 as a whole is said to be in Deep Sleep Mode.

The TC1765 implements four levels of clock gating:

1. Gated dynamically at the register
   The clock is shut off to a particular local resource in a functional module when this resource is not being used in that clock cycle. This operation is done primarily in the CPU, where unused resources are easily identified and controlled in each clock cycle.
2. Gated dynamically at the functional unit (Idle Mode)
   The clock is shut off at the functional unit boundary when the unit has nothing useful to do. This operation is done primarily in the CPU. For the CPU, idle mode is controlled via software.
3. Gated statically at each functional unit (Sleep Mode)
   Software can send a global sleep request to individual functional units requesting that they enter Sleep Mode. Software must determine when conditions are such that entering Sleep Mode is appropriate. The individual units can be programmed to ignore or respond to this signal. If programmed to respond, units will first complete pending operations, then will shut off their own clocks according to their own criteria.
4. Gated at the clock source (Deep Sleep Mode)
   The PLL and oscillator are shut off, thereby gating the clock to all functional units. The system can only be restored to operation by receiving a power-on reset signal from the $\overline{\text{PORST}}$ pin or a non-maskable interrupt signal from the $\overline{\text{NMI}}$ pin. Entering Deep Sleep Mode is under software control. Software must determine when conditions are such that entering Deep Sleep Mode is appropriate.

3.2.1 Clock Control

The functionality of the clock control registers varies depending on the functional unit being controlled. The clock for the CPU is controlled by the CPU hardware itself. The clock is switched off to the CPU automatically during Idle and Sleep Mode.

The FPI Bus clock has no clock control feature. It always runs at the system clock frequency f_{SYS}. However, the FPI Bus is so designed that no signal lines are switching when there is no activity on the bus. Hence, power consumption of the FPI Bus is minimized by design.

Similar operation applies to the CPU interrupt system. It runs with the system clock frequency f_{SYS}, however, signal lines do only change when activity, such as an arbitration round, is required.

The on-chip peripheral units of the TC1765, including the ASC0, ASC1, SSC0, SSC1, CAN, GPTU, GPTA, ADC0/ADC1, and the System Timer (STM) each have dedicated clock control registers. The generic name of these registers is given in this chapter as CLC. All clock control registers have the same bit field layout, however not all peripheral units implement all functions of these registers. In general, these registers control on/off state, clock frequency for Run Mode, operation in Sleep Mode, and operation during Debug Suspend Mode.

3.2.2 Module Clock Generation

As shown in **Figure 3-1** most of the of on-chip peripheral modules of the TC1765 have clock control registers implemented. The generic name of these registers is “CLC”. This section describes the general functionality of these CLC registers.

All CLC registers have basically the same bit and bit field layout. However, not all CLC register functions are implemented for each peripheral unit. **Table 3-5** defines in detail which bits and bit fields of the CLC registers are implemented for each peripheral module.

The CLC register basically controls the generation of the peripheral module clock which is derived from the system clock. The following functions for the module are associated with the CLC register:

- Peripheral clock static on/off control
- Peripheral clock frequency in Run Mode
- Peripheral clock frequency/behavior in Sleep Mode
- Operation during Debug Suspend Mode

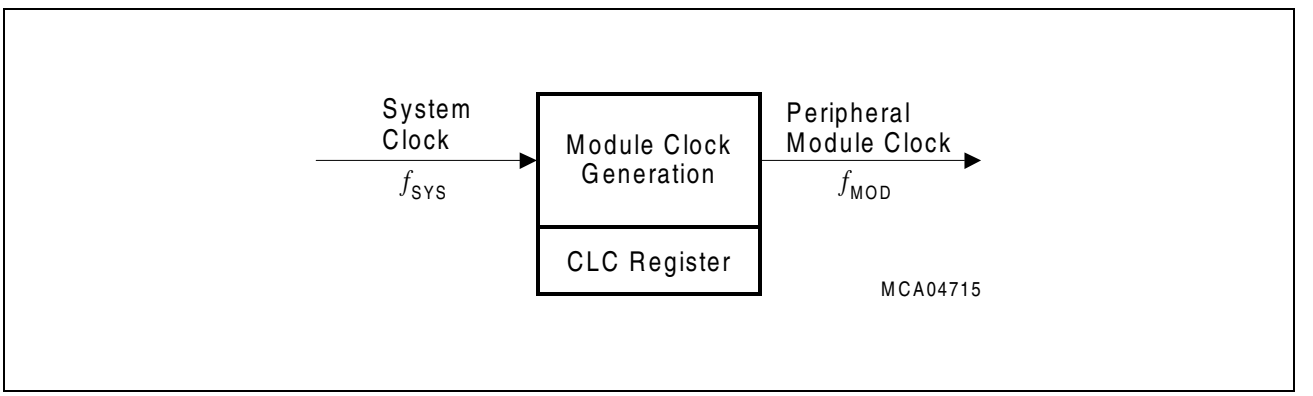
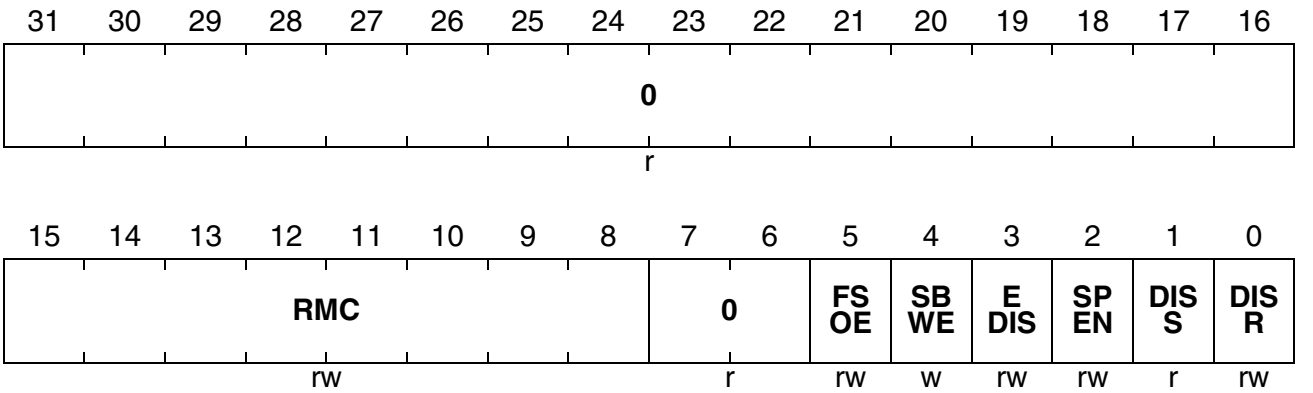


Figure 3-4 Module Clock Generation

3.2.3 Clock Control Registers

MOD_CLC
Clock Control Register **Reset Value: Module Specific**



Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module. 0 Module disable is not requested 1 Module disable is requested
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module 0 Module is enabled 1 Module is disabled
SPEN	2	rw	**Module Suspend Enable** Used for enabling the suspend mode. 0 Module cannot be suspended (suspend is disabled). 1 Module can be suspended (suspend is enabled). This bit is writable only if SBWE is set to 1 during the same write operation.
EDIS	3	rw	**Sleep Mode Enable Control** Used for module sleep mode control. 0 Sleep mode request is regarded. Module is enabled to go into sleep mode. 1 Sleep mode request is disregarded: Sleep mode cannot be entered on a request.

Field	Bits	Type	Description
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Defines whether SPEN and FSOE are write protected. 0 Bits SPEN and FSOE are write protected 1 Bits SPEN and FSOE are overwritten by respective value of SPEN or FSOE This bit is a write only bit. The value written to this bit is not stored. Reading this bit returns always 0.
FSOE	5	rw	**Fast Switch Off Enable** Used for fast clock switch off in OCDS suspend mode. 0 Clock switch off in OCDS suspend mode via Disable Control Feature (Secure Clock Switch Off) 1 Fast clock switch off in OCDS suspend mode This is writable only if SBWE is set to 1 during the same write operation.
RMC	[15:8]	rw	**8-Bit Clock Divider Value in Run Mode** Max. 8-bit divider value If RMC is set to 0 the module is disabled.
0	7, 6, [31:16]	r	**Reserved**; returns 0 if read; should be written with 0;

Module Enable/Disable Control

If a module is not used at all by an application, it can be completely shut off by setting bit DISR in its clock control register. For peripheral modules with a run mode clock divider field RMC, a second option to completely switch off the module is to set bit field RMC to 00_H. This also disables the module's operation.

The status bit DISS always indicates whether a module is currently switched off (DISS = 1) or switched on (DISS = 0). With a few exceptions (STM_CLC, EBU_CLC), the default state of a peripheral module after reset is "module disabled" with DISS set.

Write operations to the registers of disabled modules are not allowed. However, the CLC of a disabled module can be written. An attempt to write to any of the other writable registers of a disabled module except CLC will cause the Bus Control Unit (BCU) to generate a bus error.

A read operation of registers of a disabled module (except ADC0/ADC1 and CAN) is allowed and does not generate a bus error.

When a disabled module is switched on by writing an appropriate value to its MOD_CLC register, status bit DISS changes from 1 to 0. During the phase, where the module becomes active any write access to corresponding module registers (when DISS is still set) will generate a bus error. Therefore, when enabling a disabled module, application

software should check after activation of the module once whether DISS is already reset, before a module register will be written to.

Note: A read access occurring while a module is disabled is treated as a normal read access. This means, if a module register or a bit of it is cleared as a side-effect of a read access of an enabled module, it will not be cleared by this read access while the module is disabled.

Sleep Mode Control

The EDIS bit in the CLC register controls whether a module is stopped during sleep mode or not. If EDIS is 0 (default after reset), a sleep mode request can be recognized by the module and, when received, its clock is shut off.

If EDIS is set to 1, a sleep mode request is disregarded by the module and the module continues its operation.

Debug Suspend Mode Control

During emulation and debugging of TC1765 applications, the execution of an application program can be suspended. When an application is suspended, normal operation of the application's program is halted, and the TC1765 begins (or resumes) executing a special debug monitor program. When the application is suspended, a suspend signal is generated by the TC1765 and sent to all modules. If bit SPEN is set to 1, the operation of the peripheral module is stopped when the suspend signal is asserted. If SPEN is set to 0, the module does not react to the suspend signal but continues its normal operation. This feature allows each peripheral module to be adapted to the unique requirements of the application being debugged. Setting SPEN bits is usually performed by a debugger.

This feature is necessary because application requirements typically determine whether on-chip modules should be stopped or left running when an application is suspended for debugging. For example, a peripheral unit that is controlling the motion of an external device through motors in most cases must not be stopped so as to prevent damage of the external device due to the loss of control through the peripheral. On the other hand, it makes sense to stop the system timer while the debugger is actively controlling the chip because it should only count the time when the user's application is running.

Note that it is never appropriate for application software to set the SPEN bit. The debug suspend mode should only be set by a debug software. To guard against application software accidently setting SPEN, bit SPEN is specially protected by the mask bit SBWE. The SPEN bit can only be written if, during the same write operation, SBWE is set, too. Application software should never set SBWE to 1. In this way, user software can not accidentally alter the value of the SPEN bit that has been set by a debugger.

Note: The operation of the Watchdog Timer is always automatically stopped in debug suspend mode.

Entering Disabled Mode

Software can request that a peripheral unit shall be put into Disabled Mode by setting DISR. A module will also be put into Disabled Mode if the sleep mode is requested and the module is configured to allow Sleep Mode.

In Secure Shut-off Mode, a module first finishes any operation in progress, then proceeds with an orderly shut down. When all sub-components of the module are ready to be shut down, the module signals its clock control unit, which turns off the clock to this peripheral unit, that it is now ready for shut down. The status bit DISS is updated by the peripheral unit accordingly.

The kernel logic of the peripheral unit and its FPI Bus interface must both perform shut-down operations before the clock can be shut off in Secure Shut-off Mode. This is performed as follows. The peripheral module's FPI Bus interface provides an internal acknowledge signal as soon as any current bus interface operation is finished. For example, if there is a DMA write access to a peripheral in progress when a disable request is detected, the access will be terminated correctly. Similarly, the peripheral's kernel provides an internal acknowledge signal when it has entered a stable state. The clock control unit for that peripheral module shuts off the module's clock when it receives both acknowledge signals.

During emulation and debugging, it may be necessary to monitor the instantaneous state of the machine - including all or most of its modules - at the moment a software breakpoint is reached. In such cases, it may not be desired that the kernel of a module finish whatever transaction is in progress before stopping, because that might cause important states in this module to be lost. Fast Shut-off Mode, controlled by bit FSOE, is available for this situation.

If FSOE = 0, modules are stopped as described above. This is called Secure Shut-off Mode. The module kernel is allowed to finish whatever operation is in progress. The clock to the unit is then shut off if both the bus interface and the module kernel have finished their current activity. If Fast Shut-off Mode is selected (FSOE = 1), clock generation to the unit is stopped as soon as any outstanding bus interface operation is finished. The clock control unit does not wait until the kernel has finished its transaction. This option stops the unit's clock as fast as possible, and the state of the unit will be the closest possible to the time of the occurrence of the software breakpoint.

Note: The Fast Shut-off Mode is the only shut down operating mode available in the TC1765, regardless of the state of the FSOE bit.

Whether Secure Shut-off Mode or Fast Shut-off Mode is required depends on the application, the needs of the debugger, and the type of unit. For example, the analog-to-digital converter might allow the converter to finish a running analog conversion before it can be suspended. Otherwise the conversion might be corrupted and a wrong value could be produced when Debug Suspend Mode is exited and the unit is enabled again. This would affect further emulation and debugging of the application's program.

On the other hand, if a problem is observed to relate to the operation of the external analog-to-digital converter itself, it might be necessary to stop the unit as fast as possible in order to monitor its current instantaneous state. To do this, the Fast Shut-off Mode option would be selected. Although proper continuation of the application's program might not be possible after such a step, this would most likely not matter in such a case.

Note that it is never appropriate for application software to set the FSOE bit. Fast Shut-off Mode should only be set by debug software. To guard against application software accidently setting FSOE, bit FSOE is specially protected by the mask bit SBWE. The SPEN bit can only be written if, during the same write operation, SBWE is set, too. Application software should never set SBWE to 1. In this way, user software can not accidentally alter the value of the FSOE bit. Note that this is the same guard mechanism used for the SPEN bit. In this way, user software can not accidentally alter the value of the FSOE bit.

Module Clock Divider Control

Most of the TC1765 peripheral modules have an 8-bit or 2-bit control field in their CLC registers for Run Mode clock control (RMC). The clock divider circuit is located in the bus interface of these peripheral modules.

A value of 00_H in RMC disables the clock signals to these modules (module clock is switched off). If RMC is not equal to 00_H, the module clock for a unit is generated

$$f_{MOD} = f_{SYS} / RMC_{MOD} \quad [3.5]$$

where "MOD" stands for the module name and "RMC_{MOD}" is the content of its CLC register RMC field with a range of 1..255.

Note: The number of module clock cycles (wait states) which are required for a "destructive read" access (means: flags/bits are set/reset by a read access) to a module register of a peripheral unit depends on the selected module clock frequency.
Therefore, a slower module clock (selected via bit field RMC in the CLC register) results in a longer read access time for peripheral units with "destructive read" access (e.g. ASC, SSC).

3.2.4 CLC Register Implementations

Table 3-5 shows which of the CLC register bits/bit fields is implemented for each peripheral module in the TC1765.

Table 3-5 CLC Registers in the TC1765

Register	Module		DISS, DISR, Bit [1:0]	SPEN Bit 2	EDIS Bit 3	SBWE Bit 4	FSOE Bit 5	RMC Bit [15:8]
	Name	State after Reset						
CAN_CLC	CAN	disabled	■	■	■	■	–	■
ADC0_CLC	ADC0 ADC1	disabled	■	■	■	■	■	■
SSC0_CLC	SSC0	disabled	■	■	■	■	■	■
SSC1_CLC	SSC1	disabled	■	■	■	■	■	■
ASC0_CLC	ASC0	disabled	■	■	■	■	■	■
ASC1_CLC	ASC1	disabled	■	■	■	■	■	■
GPTU_CLC	GPTU	disabled	■	■	■	■	■	■
GPTA_CLC	GPTA	disabled	■	■	■	■	■	■
EBU_CLC	EBU	enabled	■	–	–	–	–	–
STM_CLC	STM	enabled	■	–	–	–	–	–
DMA_CLC	DMA	disabled	■	■	■	■	■	■
PLL_CLC	PLL	enabled	completely different bit definitions (see Section 3.1.5)					

Note: The ports of the TC1765 don't provide CLC registers.

4 System Control Unit

4.1 Overview

The System Control Unit (SCU) of the TC1765 handles the system control tasks All these system functions are tightly coupled, thus, they are conveniently handled by one unit, the SCU. The system tasks of the SCU are:

- Reset Control (described in **Chapter 5**)
  - Generation of all internal reset signals
  - Generation of external $\overline{\text{HDRST}}$ reset signal
- PLL Control (described in **Chapter 3**)
  - PLL_CLC Clock Control Register
- Power Management Control (described in **Chapter 6**)
  - Enabling of several power-down modes
  - Control of the PLL in power-down modes
- Watchdog Timer (described in **Chapter 18**)
- Trace Control and Trace Status indication
- Pull-up/pull-down I/O control
- Device Identification

This chapter describes the last three tasks in this feature list. The other tasks are described in other chapters of this document, as indicated.

4.2 Registers Overview

The basic SCU registers can be divided into three types, as shown in **Figure 4-1**. **Table 4-1** provides the long name, offset address, and location details for each of the basic registers.

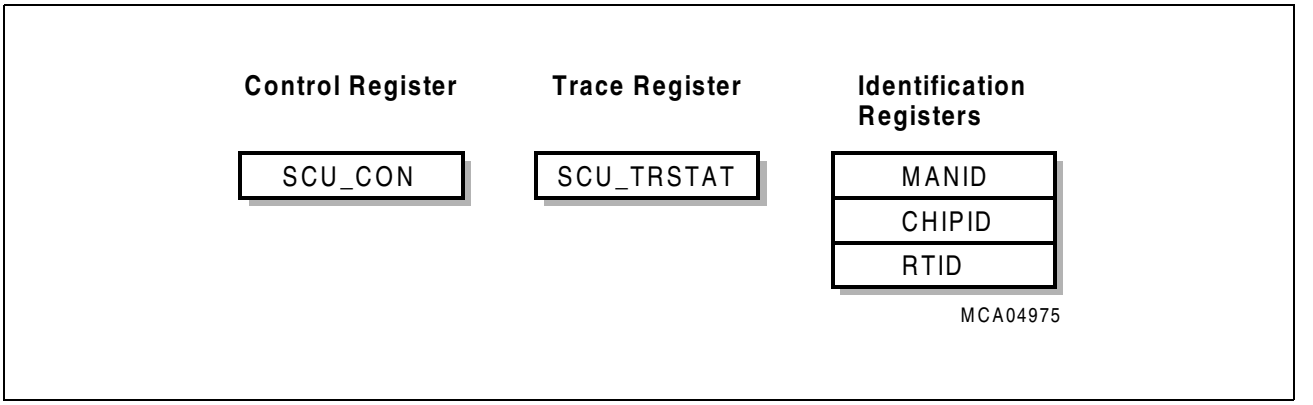


Figure 4-1 SCU Registers

Table 4-1 SCU Registers

Register Short Name	Register Long Name	Offset Address	Description see
SCU_CON	SCU Control Register	0050_H	Page 4-3
SCU_TRSTAT	SCU Trace Status Register	0054_H	Page 4-8
MANID	Manufacturer Identification Register	0070_H	Page 4-9
CHIPID	Chip Identification Register	0074_H	Page 4-10
RTID	Redesign Tracing Identification Register	0078_H	Page 4-11

In the TC1765, the registers of the SCU are located in the following address range:

- Module Base Address: F000 0000_H
  Module End Address: F000 $00FF_H$
- Absolute Register Address = Module Base Address + Offset Address
  (offset addresses see **Table 4-1**)

4.3 SCU Control Register

The bits in the SCU Control Register SCU_CON are used for:

- Control of pull-up/pull-down resistors during power-down mode
- External instruction fetch path selection and enable control
- Clock output ECOUT and CLKOUT enable control
- EBU enable control
- EBU Pull-up/pull-down resistors control

SCU_CON
SCU Control Register **Reset Value: 00F0 0030$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				0					1		EBUC PEN	CLKOUT DIS	CANTM EN	EBUD PEN	EBUA PEN
rw				r					rw		rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EBU EN	EC OUT DIS	0		0		EN SW IF	EXT IF	DIS PR DPD	0	TCU BOU TEN	TCU BIN EN	DMA BOU TEN	0	ET SEL	ET EN
rw	rw	rw		r		rw	rw	rw	r	rw	rw	rw	r	rw	rw

Field	Bits	Type	Description
ETEN	0	rw	**Emulation Trace Enable** 0 Emulation trace on OCDS-2 trace port TP[15:0] disabled (default after reset) 1 Emulation trace on OCDS-2 trace port TP[15:0] enabled
ETSEL	1	rw	**Emulation Trace Select** 0 CPU trace selected for (default after reset) 1 DMA trace selected
DMABOUTEN	3	rw	**DMA Break Output Enable** 0 $\overline{\text{BRKOUT}}$ signal disabled for DMA (default after reset) 1 $\overline{\text{BRKOUT}}$ signal enabled for DMA break out function
TCUBINEN	4	rw	**TCU Break Input Enable** 0 $\overline{\text{BRKIN}}$ signal disabled for TCU 1 $\overline{\text{BRKIN}}$ signal enabled for TCU break in function (default after reset)

Field	Bits	Type	Description
TCUBOUTEN	5	rw	**TCU Break Output Enable** 0 $\overline{\text{BRKOUT}}$ signal disabled for TCU 1 $\overline{\text{BRKOUT}}$ signal enabled for TCU break out function (default after reset)
DISPRDPD	7	rw	**Disable Pull-up/Pull-down Resistors During Power-Down Mode** 0 Pull-up/pull-down resistors are enabled during power-down mode (default after reset). 1 Pull-up/pull-down resistors are disabled during power-down mode.
EXTIF	8	rw	**External Instruction Fetch Path Select** Selects the path for external instruction fetches from the PMU (see also bit ENSWIF). 0 Instruction fetches from PMU via FPI Bus and EBU selected (default after reset) 1 Instruction fetches from PMU directly via EBU selected

EXTIF	ENSWIF	Description
X	0	Switch of external instruction fetch path selection disabled
0	1	Instruction fetch via FPI Bus
1	1	Instruction fetch direct

ENSWIF	9	rw	**Enable Switch of Instruction Fetch Path** Enables or disables switching of the instruction fetch path selection (see bit EXTIF). 0 Disable instruction fetch path switching (default after reset) 1 Enable instruction fetch path switching
ECOUTDIS	14	rw	**ECOUT Disable Control** 0 Clock signal at pin ECOUT is enabled (default after reset) 1 Clock signal at pin ECOUT is disabled. In this case, ECOUT drives a low level. *Note: If ECOUTDIS = 0, external burst mode instruction fetches are not possible.*

Field	Bits	Type	Description
EBUEN	15	rw	**EBU Enable** 0 No effect (default after reset) 1 Enables the EBU, when it is disabled (booting with CFG[2:0] = $10X_B$). Setting EBUEN has no effect when the EBU is enabled. The value last written to this bit is always read. Bit EBUEN overwrites the boot configuration selected through pins CFG[2:0]. Note that the EBU cannot be disabled anymore (after it has been enabled) except through a boot operation with combinations of CFG[2:0] except CFG[2:0] = $10X_B$.
EBUAPEN	16	rw	**EBU Address Lines Pull-up Enable** 0 Pull-up resistors at A[23:0] are disabled (default after reset) 1 Pull-up resistors at A[23:0] are enabled
EBUDPEN	17	rw	**EBU Data Lines Pull-up Enable** 0 Pull-up resistors at D[31:0] are disabled (default after reset) 1 Pull-up resistors at D[31:0] are enabled
CANTMEN	18	rw	**CAN Test Mode Enable** This bit enables the MSIST test mode of the TwinCAN module. 0 MSIST test mode of Twin CAN module disabled (default after reset) 1 MSIST test mode of TwinCAN module enabled *Note: This bit is for test purposes only! A user program must always write a 0 to this bit. Otherwise unpredictable results may occur.*
CLKOUTDIS	19	rw	**CLKOUT Disable Control**[1)] 0 Clock signal at pin CLKOUT is enabled. (default after reset) 1 Clock signal at pin CLKOUT is disabled. In this case, CLKOUT drives a low level.
EBUCPEN	20	rw	**EBU Control Lines Pull-up Enable** 0 Pull-up resistors at the EBU control lines are disabled 1 Pull-up resistors at the EBU control lines are enabled (default after reset)

Field	Bits	Type	Description
1	[23:21]	rw	**Reserved**; bits with no function; writing to these bits stores the value which is written; default after reset is 1.
0	13, 31	rw	**Reserved**; bit with no function; writing to this bit stores the value which is written; default after reset is 0.
0	2, 6, [12:10], [30:24]	r	**Reserved**; read as 0; writing to these bits has no effect.

1) For significantly improving the overall EMC behavior of the TC1765, the CLKOUT signal should be disabled after reset if it is not used in the application (CLKOUTDIS = 1).

4.4 Trace Control

This part of the SCU controls the interconnections of the TC1765T Trace Port with the trace interfaces of the Trace Control Unit (TCU) and the and DMA Controller.

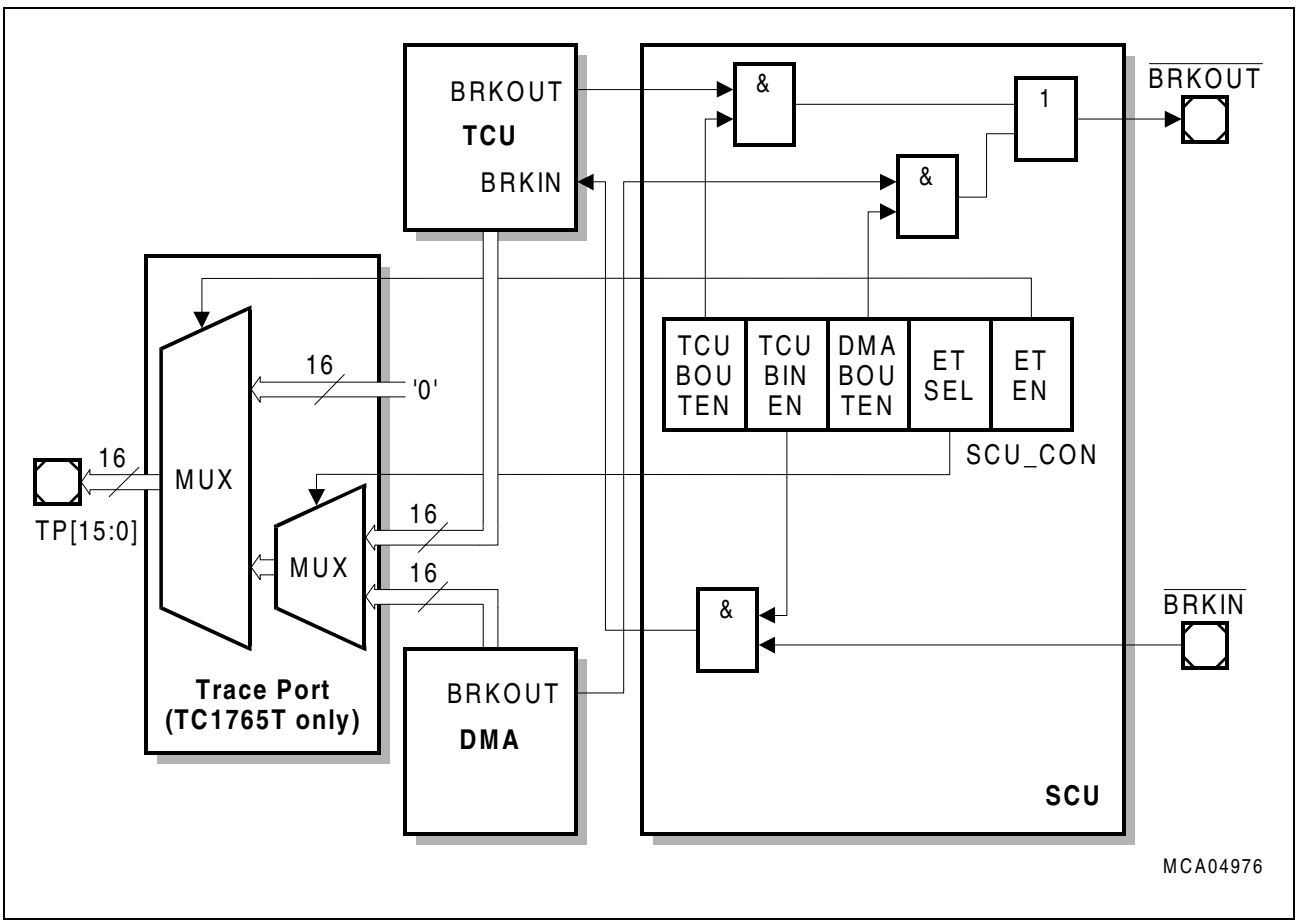
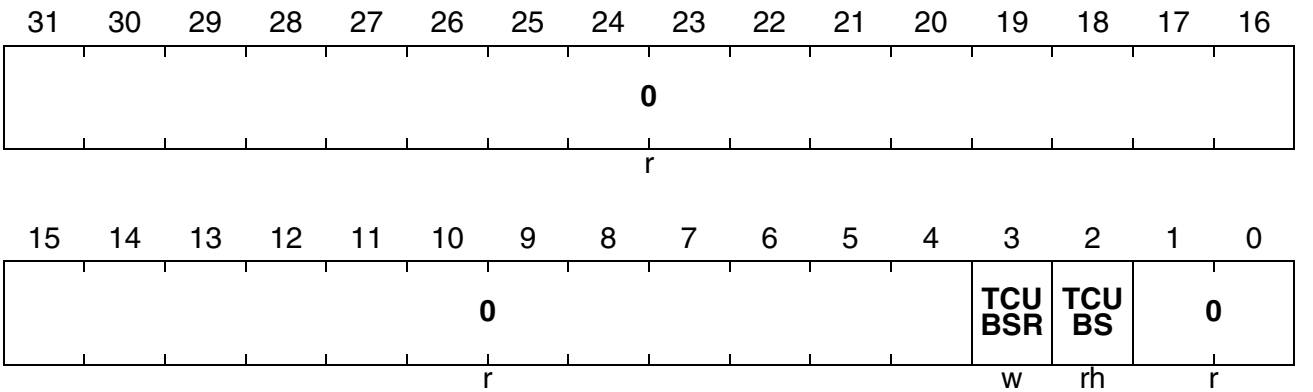


Figure 4-2 Trace Control within the SCU

*Note: The trace features of the TC1765 are described in detail in **Chapter 19** of this User's Manual.*

4.5 Trace Status Register

SCU_TRSTA
SCU Trace Status Register **Reset Value: 0000 0000$_H$**

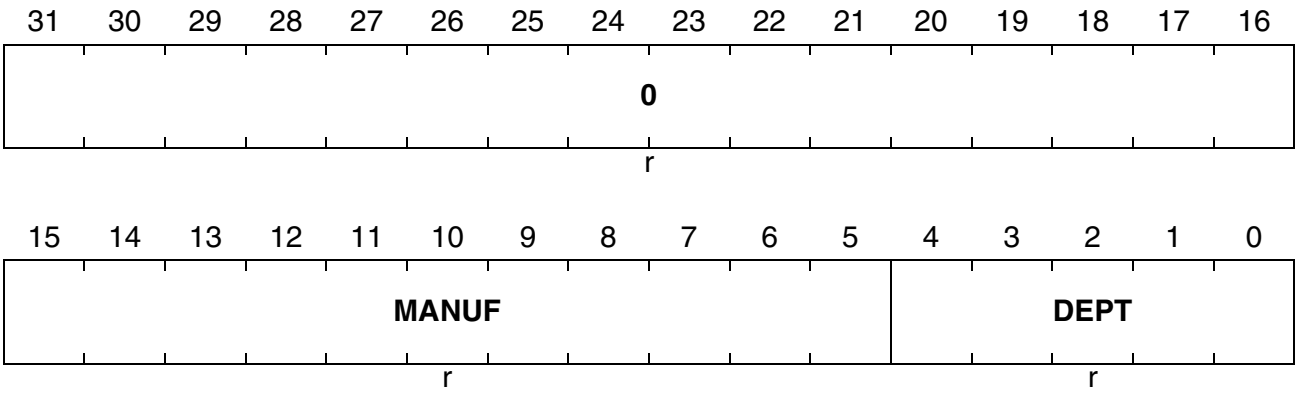


Field	Bits	Type	Description
TCUBS	2	rh	**TCU Break Status** TCUBS is set with the falling edge of the $\overline{\text{BRKIN}}$.
TCUBSR	3	w	**TCU Break Status Reset** 0 No operation 1 Reset TCUBS flag Bit is always read as 0.
0	[1:0], [31:4]	r	**Reserved**; read as 0; should be written with 0.

4.6 Identification Registers

The SCU includes four identification register: one for the SCU module identification and three for device identification.

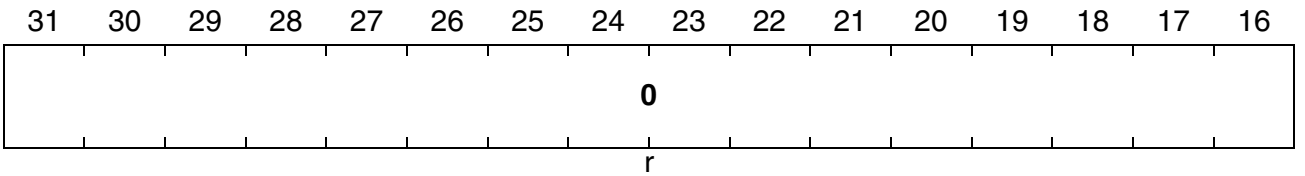
MANID
Manufacturer Identification Register **Reset Value: 0000 1820$_H$**

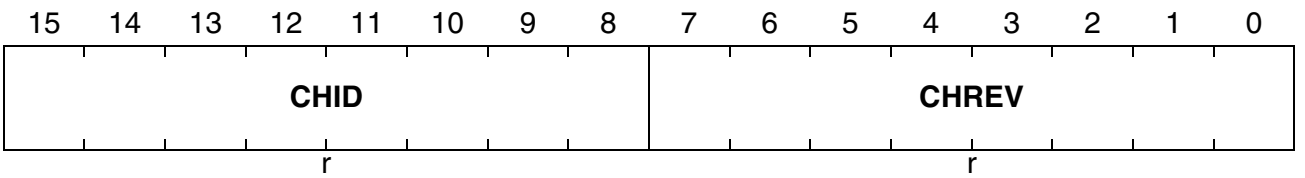


Field	Bits	Type	Description
DEPT	[4:0]	r	**Department Identification Number** = 00_H: indicates the department AI MC within Infineon Technologies.
MANUF	[15:5]	r	**Manufacturer Identification Number** This is a JEDEC normalized manufacturer code. MANUF = $C1_H$ for Infineon Technologies.
0	[31:16]	r	**Reserved**; read as 0.

CHIPID
Chip Identification Register

Reset Value: 0000 8601$_H$

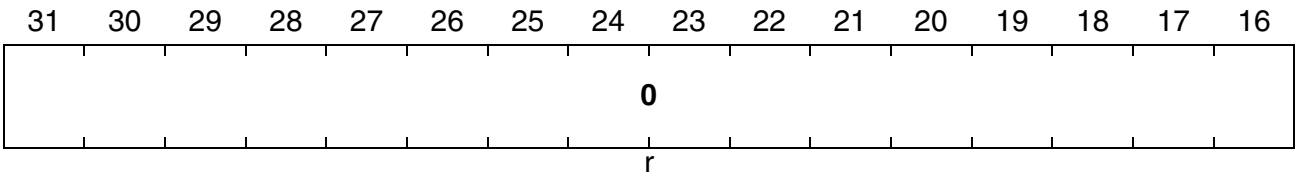


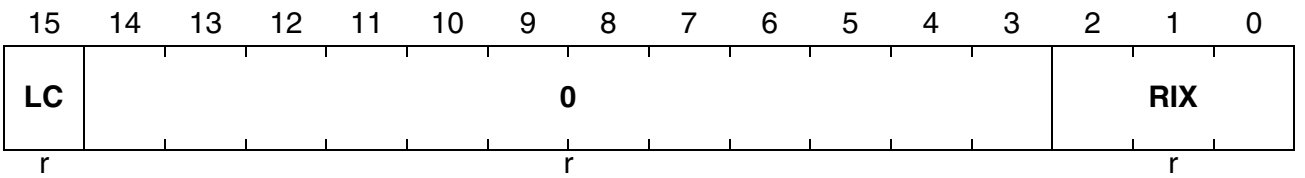


Field	Bits	Type	Description
CHREV	[7:0]	r	**Chip Revision Number** 01_H = first revision
CHID	[15:8]	r	**Chip Identification Number** 86_H = TC1765
0	[31:16]	r	**Reserved**; read as 0.

RTID
Redesign Tracing Identification Register

Reset Value: 0000 0000$_H$





Field	Bits	Type	Description
RIX	[2:0]	r	**Redesign Index** 0_H Original revision 1_H-7_H Modified revisions
LC	15	r	**Laser Correction Flag** 0 No laser correction 1 Laser correction
0	[14:3], [31:16]	r	**Reserved**; read as 0.

5 Reset and Boot Operation

This chapter describes the conditions under which the TC1765 will be reset, the reset and boot operations, and the available boot options.

5.1 Overview

When the TC1765 device is first powered up, several boot parameters must be defined to enable proper start operation of the device. Two such parameters are the operation of the PLL and the start location of the code. To accomplish parameter definition, the device has a separate Power-On Reset ($\overline{\text{PORST}}$) pin and a number of configuration pins that are sampled during the power-on reset sequence. At the end of this sequence, the sampled values are latched, and cannot be modified until the next power-on reset. This guarantees stable conditions during the normal operation of the device.

There are two ways to reset the device while it is operating: a hardware reset or a software reset. For reset causes coming from the external world, a reset input pin, $\overline{\text{HDRST}}$, is provided. If software detects conditions which require the device to be reset, a software reset can be performed by writing to a special register, the Reset Request (RST_REQ) register.

The Watchdog Timer (WDT) module is also capable of resetting the device if it detects a malfunction in the system. If the WDT is not serviced correctly and/or in time, it first generates an NMI request to the CPU (this allows the CPU to gather debug information), and then resets the device after a predefined time-out period.

Another type of reset which needs to be detected in many applications is a reset while the device is in Deep Sleep mode (Wake-Up reset). This makes it possible to distinguish a wake-up reset from a power-on reset. For a power-on reset, the contents of the memories are undefined; but, the memory contents are well defined after a wake-up reset from deep sleep.

After a reset has been executed, the Reset Status (RST_SR) register provides information on the type of the last reset and the selected boot configuration.

The external reset pin, $\overline{\text{HDRST}}$, has a double-function. It serves as a reset input from the external world to reset the device, and it serves as a reset output to the external world to indicate that the device has executed a reset. For this purpose, pin $\overline{\text{HDRST}}$ is implemented as a bidirectional open-drain pin with an internal weak pull-up device.

The boot configuration information required by the device to perform the desired start operation after a power-up reset includes the frequency selections for the PLL, the start location for the code execution, and the activation of special modes. Some of the special modes include: enabling the on-chip debugging features or placing the pins of the chip into a high-impedance mode. This information is supplied to the chip via a number of dedicated input pins which are sampled and latched with a power-on reset. However, the software reset provides the special option to alter these parameters (except for the PLL configuration) to allow a different start configuration after the software reset has finished.

5.2 Reset Registers

The two reset registers are shown in **Figure 5-1**. The long name, offset address, and location of detailed information are provided in **Table 5-1**.

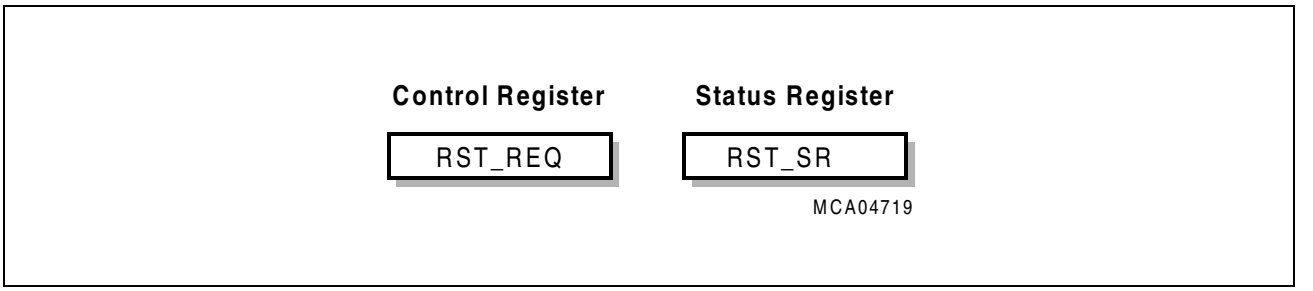


Figure 5-1 Reset Registers

Table 5-1 Reset Registers

Register Short Name	Register Long Name	Offset Address	Description see
RST_REQ	Reset Request Register	0010_H	Page 5-5
RST_SR	Reset Status Register	0014_H	Page 5-3

In the TC1765, the reset registers are located in the address range of the SCU.

- Module Base Address. F000 0000_H
  Module End Address. F000 $00FF_H$
- Absolute Register Address = Module Base Address + Offset Address
  (offset addresses see **Table 5-1**)

5.2.1 Reset Status Register (RST_SR)

After a reset, the Reset Status Register RST_SR indicates the type of reset that occurred and indicates which parts of the TC1765 were affected by the reset. It also holds the state of the boot configuration pins that are latched at power-on reset. Register RST_SR is a read-only register.

RST_SR
Reset Status Register

Power-On Reset Value: 0000 1000 00UU 1UUU 0000 0000 0000 0111_B
Hardware Reset Value: 0001 0000 00UU 1UUU 0000 0000 0000 0010_B
Software Reset Value: 0010 0000 00UU 1UUU 0000 0000 0000 $0UUU_B$
Watchdog Timer Reset Value: 0100 0000 00UU 1UUU 0000 0000 0000 0101_B
Power-Down Wake-up Reset Value: 1000 0000 00UU 1UUU 0000 0000 0000 0011_B

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PWD RST	WDT RST	SFT RST	HD RST	PWO RST	0					HW BRK IN	HW OCD SE	1	HWCFG		
rh	rh	rh	rh	rh	r					rh	rh	r	rh		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													RS EXT	X	RS STM
r													rh	r	rh

Field	Bits	Type	Description
RSSTM	0	rh	**System Timer Reset Status** 0 System Timer was not reset 1 System Timer was reset
X	1	r	**Reserved;** bit has an undefined value when it is read; default after reset state depends on reset cause.
RSEXT	2	rh	**$\overline{\text{HDRST}}$ Line State during Last Reset** 0 $\overline{\text{HDRST}}$ was not activated as output by TC1765 1 $\overline{\text{HDRST}}$ was activated as output by TC1765
HWCFG	[18:16]	rh	**Boot Configuration Selection Status** Status of the configuration pins CFG[2:0] latched with power-on reset.
HWOCDSE	20	rh	**State of $\overline{\text{OCDSE}}$ Pin** Value of the OCDS enable pin latched at the end of power-on reset.

Field	Bits	Type	Description
HWBRKIN	21	rh	**State of $\overline{\text{BRKIN}}$ Pin** Value of the break input pin latched at the end of power-on reset.
PWORST	27	rh	**Power-On Reset Status Flag** 0 The last reset was not a power-on reset 1 The last reset was a power-on reset
HDRST	28	rh	**Hardware Reset Status Flag** 0 The last reset was not a hardware reset 1 The last reset was a hardware reset
SFTRST	29	rh	**Software Reset Status Flag** 0 The last reset was not a software reset 1 The last reset was a software reset
WDTRST	30	rh	**Watchdog Reset Status Flag** 0 The last reset was not a watchdog reset 1 The last reset was a watchdog reset
PWDRST	31	rh	**Power-Down/Wake-Up Reset Status Flag** 0 The last reset was not a wake-up from power-down reset 1 The last reset was a wake-up from power-down reset
0	[15:3], 19, [26:22]	r	**Reserved**; returns 0 if read.

5.2.2 Reset Request Register (RST_REQ)

The Reset Request Register RST_REQ is used to generate a software reset. Unlike the other reset types, the software reset can exclude two functions from the reset. These are the System Timer and the external reset output HDRST. In addition, it can change the boot configuration.

A software reset is invoked by any write to register RST_REQ. This register is EndInit-protected, meaning that bit WDT_CON0.ENDINIT must be set to 0 first through the password-protected access scheme for WDT_CON0. Once access is gained through the Endinit protection scheme, RST_REQ can be written, causing a software reset.

RST_REQ
Reset Request Register **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0							SWBOOT	0		SWBRKIN	SWOCDSE	0	SWCFG		
r							rw	r		rw	rw	rw	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										RSTLEN		0	RREXT	X	RRSTM
r										rw		r	rw	rw	rw

Field	Bits	Type	Description
RRSTM	0	rw	**Reset Request for the System Timer** 0 Do not reset the System Timer 1 Reset the System Timer
X	1	rw	**Reserved**; bit with no function; writing to this bit stores the value which is written; default after reset is 0.
RREXT	2	rw	**Reset Request for External Devices** 0 Do not activate reset output HDRST 1 Activate reset output HDRST

Field	Bits	Type	Description
RSTLEN	[5:4]	rw	**Reset Length Control** This bit field defines the length (duration) for a software reset or for a watchdog reset (t_{SYS} is the period of the system clock with frequency f_{SYS}). 00_B Reset duration is $16 \times t_{SYS}$. This combination is valid for hardware reset and power-on reset. 01_B Reset duration is $80 \times t_{SYS}$ 10_B Reset duration is $1040 \times t_{SYS}$ 11_B Reset duration is $1104 \times t_{SYS}$
SWCFG	[18:16]	rw	**Software Boot Configuration** A software boot configuration different from the external applied hardware configuration can be specified with these bits. The configuration encoding is equal to the CFG[2:0] encoding.
SWOCDSE	20	rw	**Software OCDS Enable Signal Boot Value** Determines the desired value for the OCDS enable input signal to be used for software boot.
SWBRKIN	21	rw	**Software Break Signal Boot Value** Determines the desired value for the break input signal to be used for software boot.
SWBOOT	24	rw	**Software Boot Configuration Selection** 0 Use the previously latched hardware configuration 1 Use the programmed software configuration
0	19	rw	**Reserved**; read as 0; should be written with 1.
0	3, [15:6], 22, 23, [31:25]	r	**Reserved**; read as 0; should be written with 0.

5.3 Reset Operations

A detailed description of each of the reset options is given in the following sections.

5.3.1 Power-On Reset

The $\overline{\text{PORST}}$ pin performs a power-on reset, also called cold reset. Driving the $\overline{\text{PORST}}$ pin low causes an asynchronous reset of the entire device. The device then enters its power-on reset sequence.

The external configuration input pins for the PLL are sampled in order to select the proper operating mode of the PLL. The PLL itself has its own power-on reset circuitry, and is not affected by any other reset condition other than a low signal transition on the $\overline{\text{PORST}}$ pin. The values of the PLL configuration pins are sampled in register PLL_CLC.

Simultaneously, the reset circuitry in the TC1765 drives the $\overline{\text{HDRST}}$ pin low, and then waits for the following two conditions to occur:

1. The system clock is active
2. Pin $\overline{\text{PORST}}$ is negated (driven high)

When both of these conditions are met the $\overline{\text{HDRST}}$ output is released and tri-stated by the TC1765. If within this tri-state phase $\overline{\text{HDRST}}$ is still driven by an external device, the TC1765 remains in the reset condition as long as $\overline{\text{HDRST}}$ is at low level. When $\overline{\text{HDRST}}$ is again pulled to high level externally, the power-on reset sequence is terminated. The power-on reset indication flag PWORST in the Reset Status Register RST_SR is set, while all other reset cause indication flags are cleared. (Fields in this register that are set include the power-on reset indication flag (PWORST), as well as the reset status flags for the System Timer (RSSTM) and the reset output pin (RSEXT).

5.3.2 External Hardware Reset

The external hardware reset pin $\overline{\text{HDRST}}$ serves as an external reset input as well as a reset output. It is an active-low, bidirectional open-drain pin with an internal weak pull-up. An active-low signal at this pin causes the chip to enter its internal hardware reset sequence at the next system clock (f_{SYS}) transition. For synchronization purposes, $\overline{\text{HDRST}}$ must be at low level at least for two system clock cycles. $\overline{\text{HDRST}}$ is not actively driven by the TC1765 during a hardware reset sequence. If other hardware is reset by the signal line connected to $\overline{\text{HDRST}}$, the $\overline{\text{HDRST}}$ driving device must meet timing requirements for this hardware. After the TC1765 has terminated its hardware reset sequence internally, flag RST_SR.HDRST is set.

Note that a hardware reset does not cause the configuration pins for the PLL and boot options to be latched. The configuration state that was latched at the end of the last power-on reset still controls these functions. Also, the PLL is not affected by an external hardware reset, but continues to operate according to its selected mode.

5.3.3 Software Reset

A software reset is invoked by writing the appropriate bits in the Reset Request Register (RST_REQ). Unlike the other forms of reset, the software reset can exclude two system functions from being reset. These are the System Timer and the external reset output $\overline{\text{HDRST}}$. Also, a software reset can change the boot configuration as a side-effect.

Excluding some system functions from a software reset offers these potential advantages:

- The System Timer can continue to clock accumulated elapsed time.
- The external components of a system can continue to operate while only the TC1765 is reset.

To perform a software reset, the Reset Request Register RST_REQ must be written. However, RST_REQ is EndInit-protected to avoid an unintentional software reset. The ENDINIT bit in the Watchdog Timer control register WDT_CON0 must be cleared via the password-protected access scheme. When this is done, a write access to RST_REQ can then be performed.

To exclude system functions from software reset, the appropriate bits in RST_REQ must be set to 0:

- Set RREXT to 0 to avoid activating the reset output $\overline{\text{HDRST}}$
- Set RRSTM to 0 to avoid resetting the System Timer

To change the boot configuration latched at the end of power-on reset, the software boot selection bit SWBOOT must be set, and the desired boot configuration must be written to bits SWBRKIN, SWOCDSE, and SWCFG[3:0].

When the software reset is terminated, bit RST_SR.SFTRST is set, indicating that the last reset was a software reset. All other reset cause indication flags are cleared. The reset status of the System Timer (RSSTM) and $\overline{\text{HDRST}}$ pin (RSEXT) are set according to the bits in RST_REQ at the time the software reset was initiated.

The PLL is not affected by a software reset; it continues to operate according to its previous mode.

Note that the boot configuration bits in the Reset Request Register RST_REQ are only used on software reset. In particular, the SWCFG bits that can be set to cause the TC1765 to boot using internal memory (if SWCFG is set to 0) are not effective on hardware boot. Regardless of the state of RST_REQ, any reset other than a software reset always uses the hardware configuration.

Bit field RST_REQ.RSTLEN defines the length of a software reset operation and with that also the length of the $\overline{\text{HDRST}}$ output signal when its activation is enabled by RSTREQ.RREXT = 1. Note, that if after the end of such a software reset operation ($\overline{\text{HDRST}}$ output released and tri-stated by the TC1765) $\overline{\text{HDRST}}$ is still driven low by an external device, the TC1765 detects an external hardware reset condition and starts executing it as described in **Section 5.3.2**.

5.3.4 Watchdog Timer Reset

A Watchdog Timer overflow or access error occurs only in response to severe and/or unknown malfunctions of the TC1765, caused by software or hardware errors. Therefore, the entire TC1765 is given a Watchdog Timer reset whenever the Watchdog Timer overflows.

Before the Watchdog Timer generates its reset, it first signals a non-maskable interrupt (NMI) and enters a time-out mode. The NMI invokes a Trap Service Routine (NMI is really a trap, not an interrupt). The trap handler can save critical state of the machine for subsequent examination of the cause of the Watchdog Timer failure. However, it is not possible to stop or terminate the Watchdog Timer's time-out mode or prevent the pending watchdog reset.

However, software can preempt the Watchdog Timer by issuing a software reset on its own. Because the cause of the system failure is presumably unknown at that time, and it is presumably uncertain which functions of the TC1765 are operating properly, it is recommended that the software reset be configured to reset all system functions including the System Timer and external reset output $\overline{\text{HDRST}}$, and to use the hardware boot configuration.

Eventually, if the NMI trap handler does not perform a software reset, or if the system is so compromised that the trap handler cannot be executed, the Watchdog Timer will cause a Watchdog Timer reset to occur at the end of its time-out mode period. The actions performed on a Watchdog Timer reset sequence are the same as are performed for an external hardware reset. At the end of the Watchdog Timer reset sequence, bits WDTRST, RSSTM, and RSEXT are set in register RST_SR. All other reset cause indication flags are cleared.

5.3.4.1 Watchdog Timer Reset Lock

When the system emerges from any reset condition, the Watchdog Timer becomes active, and, — unless prevented by initialization software — will eventually time out. Ordinarily, initialization software will configure the Watchdog Timer and commence servicing it on a regular basis to indicate that it is functioning appropriately. Should the system be malfunctioning so that initialization and service are not performed in a timely fashion, the Watchdog Timer will time out, causing a Watchdog Timer reset.

If the TC1765 system is so corrupted that it is chronically unable to service the Watchdog Timer, the danger could arise that the system would be continuously reset every time the Watchdog Timer times out. This could lead to serious system instability, and to the loss of information about the original cause of the failure.

However, the reset circuitry of the TC1765 is designed to detect a double reset condition of the Watchdog Timer. If a Watchdog induced reset occurs twice, a severe system malfunction is assumed and the TC1765 is held in reset until a power-on reset or hardware reset occurs (see also **Chapter 18**).

5.3.4.2 Deep-Sleep Wake-Up Reset

Power is still applied to the TC1765 during Deep Sleep power-management mode, which preserves the contents of the TC1765's static RAM. If Deep Sleep mode is entered appropriately, all important system state information will have been preserved in static RAM by software. The only way to terminate Deep Sleep mode is for the TC1765 to be externally reset. However, while external reset will cause the TC1765's registers to return to their default reset values, the contents of the static RAM is not affected. This can be important to the application software because initialization of the static RAM can be skipped, and data written to it before Deep Sleep mode was entered will still be valid.

If the TC1765 is in Deep Sleep mode, there are three options to awaken it:

1. A power-on reset $\overline{\text{PORST}}$
2. An external $\overline{\text{NMI}}$ event with a reset sequence
3. An external $\overline{\text{NMI}}$ event without a reset sequence

Selection between the two types of external $\overline{\text{NMI}}$ event is made via the control bit PM_CON.DSRW. The advantage of using an external $\overline{\text{NMI}}$ event without a reset sequence is that the system can be more quickly awakened.

5.3.5 State of the TC1765 after Reset

Table 5-2 lists the modules/functions and types of reset and indicates whether and how the various functions of the TC1765 are affected. A ■ indicates that a function is reset to its default state.

Table 5-2 Effect of Reset Types on TC1765 Modules/Functions

Module / Function	Wake-up Reset	Watchdog Reset	Software Reset	Hardware Reset	Power-On Reset
CPU Core	■	■	■	■	■
Peripherals (except System Timer)	■	■	■	■	■
On-Chip Static RAM (code or data)	Not affected	Not affected; contents may be unreliable	Not affected	Not affected	Not affected; contents are invalid
On-Chip Cache (see note)	■	■	■	■	■
System Timer	■	■	Optional	Not affected	■

Table 5-2 Effect of Reset Types on TC1765 Modules/Functions (cont'd)

Module / Function	Wake-up Reset	Watchdog Reset	Software Reset	Hardware Reset	Power-On Reset
Debug Unit	■	■	Optional	■	■
Oscillator / PLL	■	Not affected	Not affected	Not affected	■
External Bus Control Unit	■	■	■	■	■
External Bus Pins	Tri-stated	Tri-stated	Tri-stated	Tri-stated	Tri-stated
Port Pins	Tri-stated	Tri-stated	Tri-stated	Tri-stated	Tri-stated
Reset output pin $\overline{\text{HDRST}}$	■	■	Optional	Tri-Stated	■
Boot Configuration taken from	Latched hardware configur-ation	Latched hardware configur-ation	Optional latched hardware or software configur-ation	Latched hardware configur-ation	External pins
PLL Configuration taken from	Latched hardware configur-ation	Latched hardware configur-ation	Latched hardware configur-ation	Latched hardware configur-ation	External pins

Note: The actual data contents of the cache are not affected through a reset; however the cache tag information is cleared, resulting in an 'empty' cache.

5.4 Booting Scheme

When the TC1765 is reset, it needs to know the type of configuration required to start in after the reset sequence is finished. Internal state is usually cleared through a reset. This is especially true in the case of a power-up reset. Thus, boot configuration information needs to be applied by the external world through input pins.

Boot configuration information is required:

- for the PLL to select the proper operating mode and frequency,
- for the start location of the code execution,
- and activation of special modes and conditions.

PLL configuration is only sampled and latched with a power-on reset.

For the start of code execution and activation of special mode, the TC1765 implements two basic booting schemes: a hardware scheme which is invoked through external pins, and a software scheme in which software can determine the boot options, overriding the externally-applied options.

5.4.1 Hardware Booting Scheme

The hardware booting scheme uses the state of a number of external input pins — sampled and latched with a power-on reset — to determine the start configuration of the chip. The state of these pins is latched into the Reset Status Register RST_SR when the power-on reset signal (pin $\overline{\text{PORST}}$) is released. This hardware configuration determined through the bits HWOCDSE, HWBRKIN, and HWCFG[2:0] is used for all hardware-invoked reset options (power-on, hard, watchdog and wake-up reset).

5.4.2 Software Booting Scheme

The Reset Request Register RST_REQ, used for generating a software reset, contains five bits that have the same meaning as the corresponding five bits in the RST_SR register. On a software reset, software can choose to set a different boot configuration from the one latched with power-on reset. This option is selected through bit SWBOOT in register RST_REQ. When writing to this register, the desired values for bits SWOCDSE, SWBRKIN, and SWCFG[2:0] are written along with bit SWBOOT set to 1. This causes the device to start in the configuration selected through the software boot configuration bits in register RST_REQ instead of starting with the hardware boot configuration stored in register RST_SR.

5.4.3 Boot Options

The architecture of the TriCore booting schemes provides a number of different boot options for the start of code execution. **Table 5-3** shows the boot options available in the TC1765. Note that the signals $\overline{\text{OCDSE}}$, $\overline{\text{BRKIN}}$, and CFG[2:0] can be either the corresponding bits HWOCDSE, HWBRKIN, and HWCFG[2:0] in register RST_SR, or the software configuration bits SWOCDSE, SWBRKIN and SWCFG[2:0] in register RST_REQ.

Table 5-3 TC1765 Boot Selections

$\overline{\text{OCDSE}}$	$\overline{\text{BRKIN}}$	CFG[2:0]	Type of Boot	Boot Source	PC Start Value
1	1	000_B	Start from Boot ROM Entry Point 1	Boot ROM	$BFFFFFFC_H$
		001_B	Start from Boot ROM Entry Point 2		
		010_B	Start from Boot ROM Entry Point 3		
		011_B	Start from Boot ROM Entry Point 4		
		100_B	External memory as master directly PMU - EBU	External Memory	$A000\ 0000_H$
		101_B	External memory via PMU - FPI Bus - EBU		
		110_B	Reserved; don't use these combinations.		
		111_B			
0	1	100_B or 101_B	Go to halt with EBU enabled as master	–	–
		all other combina-tions	Go to halt with EBU disabled		
0	0	–	Go to external emulator space	–	$BE00\ 0000_H$
1	0	–	Tri-state chip (deep sleep)	–	–

5.4.4 Boot Configuration Handling

- The inputs CFG[2:0] at Port 4 are latched internally with the rising edge of $\overline{\text{PORST}}$ to guarantee a stable value during normal operation (during $\overline{\text{PORST}}$ active the latches are transparent). The latched values can only be changed by another power-on reset.
- The CFG[2:0] pins determine the hardware boot configuration after power-on reset / hardware reset. This configuration can be changed by software in conjunction with a software reset (software boot configuration).
- The boot software must read the actual software configuration (register RST_REQ) to determine how to proceed (for example: entering boot-strap loader). It is also possible to read the latched value of the configuration pins.

5.4.5 Normal Boot Options

The normal boot options are invoked when both, $\overline{\text{OCDSE}}$ and $\overline{\text{BRKIN}}$ are set to 1.

In order to access external memory, the External Bus Unit (EBU) must have information about the type and access mechanism of the external boot code memory. This information is not available through the boot configuration pins. Special actions must be taken first by the EBU in order to determine the configuration settings.

The EBU initiates a special external bus access in order to retrieve information about the external code memory. This access is performed to address A000 0004_H such that regardless of the type and characteristics of the external memory, configuration information can be read from the memory into the EBU. By examining this information, the EBU determines the exact requirements for accesses to the external memory. It then configures the control registers accordingly, and performs the first instruction fetch from address A000 0000_H.

Note: After a boot operation of the TC1765 with combinations of CFG[2:0] except CFG[2:0] = $10X_B$ the EBU is disabled and must be enabled by setting SCU_CON.EBUEN before it can be used.

5.4.6 Debug Boot Options

Debug boot options are selected if the states of the bits $\overline{\text{OCDSE}}$ and $\overline{\text{BRKIN}}$ are not both activated.

Two of the options enable emulators to take control over the TC1765. If only $\overline{\text{OCDSE}}$ is activated ($\overline{\text{OCDSE}}$ = 0), the TC1765 goes into the HALT state. External hardware emulators can then configure the TC1765 via the JTAG interface. If $\overline{\text{BRKIN}}$ and $\overline{\text{OCDSE}}$ are activated ($\overline{\text{BRKIN}}$ = $\overline{\text{OCDSE}}$ = 0), the TC1765 starts execution out of a special external memory region reserved for debugging.

After configuring the TC1765 via either of these boot options, the regular application configuration can be invoked by executing a software reset with a software boot option. By setting the software configuration bits in register RST_REQ such that the debug boot

options are deactivated, a normal boot of the TC1765 is accomplished after the software reset terminates.

Note: The state of the external $\overline{OCDSE}$ pin is also latched by other circuitry in the TC1765, enabling special debugging features if a low signal level is latched at this pin when the power-on reset ($\overline{PORST}$) signal is raised. A software boot with a normal boot configuration (that is, bit SWOCDSE = 1) does not affect this operation.

The third debug boot option places the TC1765 into a tri-state mode. All pins are deactivated, including the oscillator, and internal circuitry is held in a low-power mode. This mode can be used to connect emulator probes to a TC1765 soldered onto a board to perform testing.

5.4.7 Booting from Boot ROM

When booting from Boot ROM (see **Table 5-3**), the EBU is turned off. Therefore, the EBU must be enabled before accessing external data or code locations from PMU or DMU. The EBU is enabled by writing to SCUCON with bits EBUEN = 1 and EBUDIS = 0. Prior to this SCUCON write operation, the initialization write operation to register BUSCON0 with the required mode and timing parameters must be executed.

6 Power Management

This chapter describes the power management system for the TC1765. Topics include the internal system interfaces, external interfaces, state diagrams, and the operations of the CPU and peripherals. The Power Management State Machine (PMSM) is also described.

6.1 Power Management Overview

The TC1765 power management system allows software to configure the various processing units so that they automatically adjust to draw the minimum necessary power for the application.

As shown in **Table 6-1**, there are four power management modes:

- Run Mode
- Idle Mode
- Sleep Mode
- Deep Sleep Mode

Table 6-1 Power Management Mode Summary

Mode	Description
Run	The system is fully operational. All clocks and peripherals are enabled, as determined by software.
Idle	The CPU clock is disabled, waiting for a condition to return it to Run Mode. Idle Mode can be entered by software when the processor has no active tasks to perform. All peripherals remain powered and clocked. Processor memory is accessible to peripherals. A reset, Watchdog Timer event, a falling edge on the $\overline{\text{NMI}}$ pin, or any enabled interrupt event will return the system to Run Mode.
Sleep	The system clock continues to be distributed only to those peripherals programmed to operate in Sleep Mode. Interrupts from operating peripherals, the Watchdog Timer, a falling edge on the $\overline{\text{NMI}}$ pin, or a reset event will return the system to Run Mode. Entering this state requires an orderly shut-down controlled by the Power Management State Machine.
Deep Sleep	The system clock is shut off; only an external signal will restart the system. Entering this state requires an orderly shut-down controlled by the Power Management State Machine (PMSM).

The operation of each system component in each of these states can be configured by software. The power management modes provide flexible reduction of power consumption through a combination of techniques, including:

- Stopping the CPU clock
- Stopping the clocks of other system components individually
- Clock-speed reduction of some peripheral components individually
- Power-down of the entire system with fast restart capability

The Power Management State Machine (PMSM) controls the power management mode of all system components during Run Mode, Idle Mode, and Sleep Mode. The PMSM continues to operate in Idle Mode and Sleep Mode, even if all other system components have been disabled, so that it can re-awaken the system as needed. In Deep Sleep Mode, even the PMSM is disabled and the system must be re-awakened from an external source. This flexibility in power management provides minimum power consumption for any application.

The Power Management State Machine is implemented in the System Control Unit (SCU) module of the TC1765. Thus, it is accessible through the FPI Bus interface by any FPI Bus master.

As well as these explicit software-controlled power-saving modes, special attention has been paid in the TC1765 to provide automatic power-saving in those operating units that are currently not required or idle. To save power, these are shut off automatically until their operation is required again.

In typical operation, Idle Mode and Sleep Mode will be entered and exited frequently during the runtime of an application. For example, system software will typically cause the CPU to enter Idle Mode each time it must wait for an interrupt before continuing its tasks. In Sleep Mode and Idle Mode, wake-up is performed automatically when any enabled interrupt signal is detected or if the Watchdog Timer signals the CPU with an NMI trap.

No clock is running in a system in Deep Sleep Mode, so it cannot be awakened by an interrupt or the Watchdog Timer. It will be awakened only when it receives an external non-maskable interrupt (NMI) or reset signal, as described **Section 6.3.3**. Software must prepare the external environment of the TC1765 to cause one of these signals under the appropriate conditions before entering Deep Sleep Mode. If Deep Sleep Mode were entered unintentionally without an event of this nature first being prepared, the TC1765 might never emerge from Deep Sleep Mode. For this reason, the register used to set up Deep Sleep Mode can be changed only by way of a password-protected access mechanism (see **Section 6.3.3**).

6.2 Power Management Control Registers

The set of registers used for power management is divided between central TC1765 components and peripheral components. The PMG_CSR and the PMG_CON registers provide software control and status information for the Power Management State Machine (PMSM). There are individual clock control registers for peripheral components because the Sleep Mode behavior of each peripheral component is programmable. When entering Idle Mode and Sleep Mode, the PMSM directly controls TC1765 components such as the CPU, but indirectly controls peripheral components through their clock control registers.

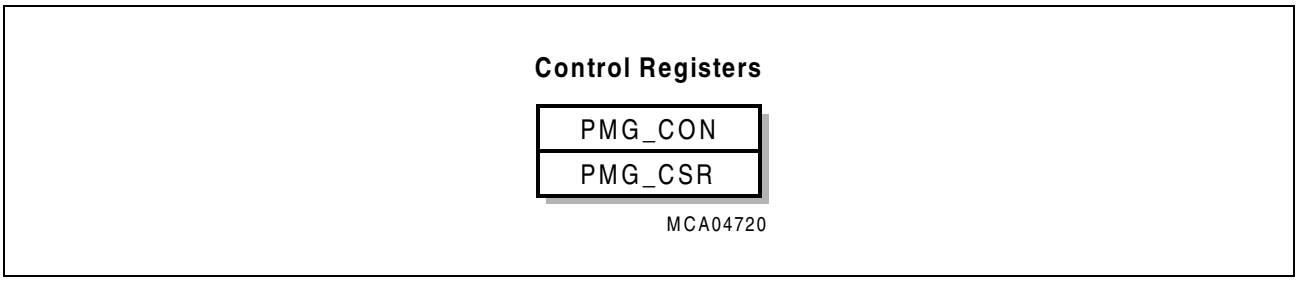


Figure 6-1 Power Management Registers

Table 6-2 Power Management Registers

Register Short Name	Register Long Name	Offset Address	Description see
PMG_CON	Power Management Control Register	0030_H	Page 6-4
PMG_CSR	Power Management Control and Status Register	0034_H	Page 6-5

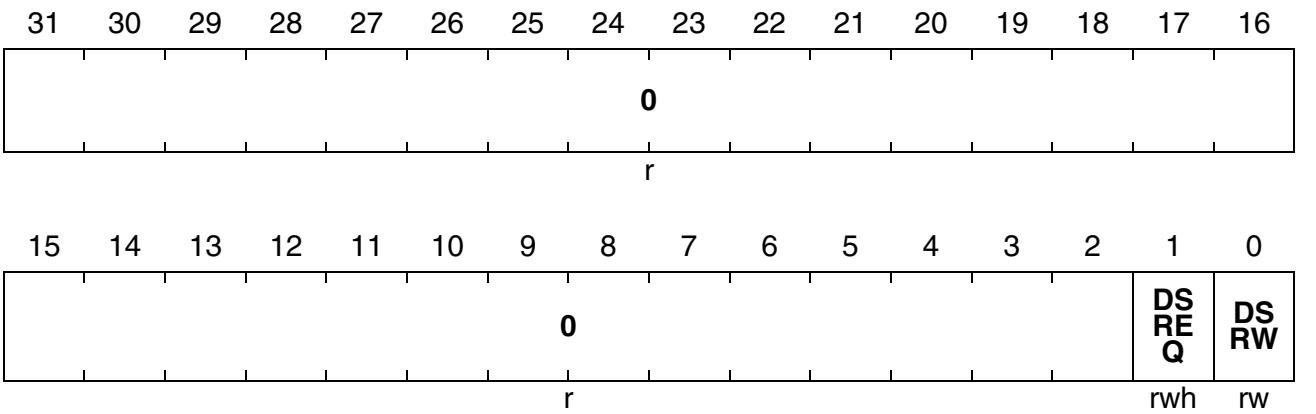
In the TC1765, the reset registers are located in the address range of the SCU:

- Module Base Address. F000 0000_H
  Module End Address. F000 $00FF_H$
- Absolute Register Address = Module Base Address + Offset Address
  (offset addresses see Table 6-2)

6.2.1 Power Management Control Register PMG_CON

The Power Management Control Register PMG_CON is used to request Deep Sleep Mode. This register is specially protected to avoid unintentional invocation of Deep Sleep Mode.

PMG_CON
Power Management Control Register **Reset Value: 0000 0001$_H$**



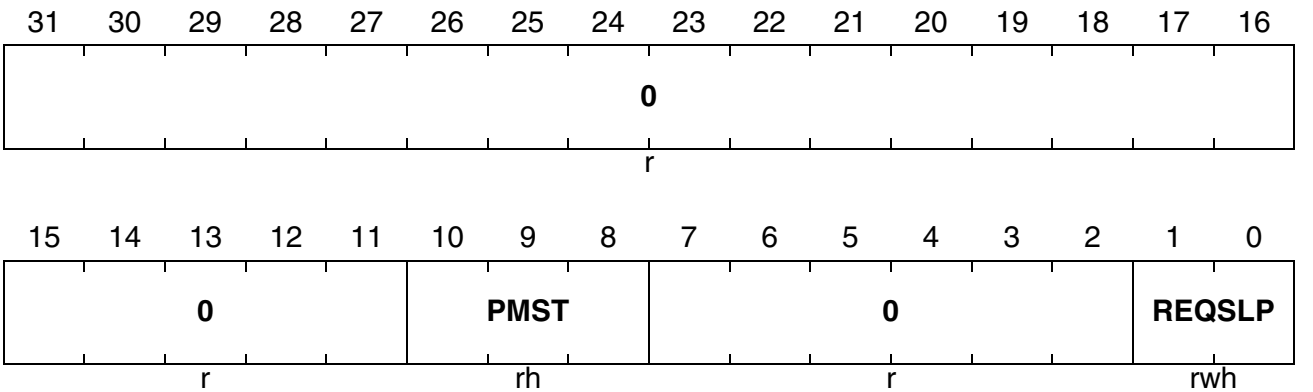
Field	Bit	Type	Function
DSRW	0	rw	**Reset On Wake-Up From Deep Sleep** Wake-up from deep sleep can be caused by either a power-on reset or through a low level at the $\overline{\text{NMI}}$ pin. The state of DSRW determines whether a full internal hardware reset should be performed on exit from deep sleep. 0 No internal reset will be performed on exit from deep sleep 1 An internal hardware reset will be performed on exit from deep sleep
DSREQ	1	rwh	**Deep Sleep Request Bit** 0 Normal Mode 1 Deep Sleep Mode requested Bit is reset by hardware on wake-up from deep sleep mode.
0	[31:2]	r	**Reserved**; read as 0; should be written with 0.

Note: The PMG_CON register is specially protected to avoid unintentional invocation of Deep Sleep Mode. In order to write to PMG_CON.DSREQ, the WDT_CON0.ENDINIT bit must be set to 0 through a password-protected access mechanism. WDT_CON0.ENDINIT must then be set to 1 to make the changed value of DSREQ become effective.

6.2.2 Power Management Control and Status Register PMG_CSR

The Power Management Control and Status Register PMG_CSR stores Idle Mode and Sleep Mode request bits. It also shows the status of the Power Management State Machine. Its fields are described below.

PMG_CSR
Power Management Control and Status Register **Reset Value: 0000 0100$_H$**



Field	Bit	Type	Function
REQSLP	[1:0]	rwh	**Idle Mode and Sleep Mode Request Bits** 00_B Normal Run Mode 01_B Request Idle Mode 10_B Request Sleep Mode 11_B Reserved; do not use this combination; In Idle Mode, Sleep Mode, or Deep Sleep Mode, these bits are cleared in response to an enabled interrupt, a wake-up from Deep Sleep Mode via the $\overline{\text{NMI}}$ pin or $\overline{\text{PORST}}$ pin, or when bit 15 of the Watchdog Timer count register (the WDT_SR.TIM[15] bit) changes from 0 to 1.
PMST	[10:8]	rh	**Power Management State Machine Status** 000_B Waiting for PLL Lock condition 001_B Normal Run Mode 010_B Idle Mode requested 011_B Idle Mode acknowledged 100_B Sleep Mode 101_B Deep Sleep Mode 110_B Undefined, reserved 111_B Undefined, reserved
0	[7:2], [31:11]	r	**Reserved**; read as 0; should be written with 0.

6.3 Power Management Modes

This section describes power management modes, their operations, and how power management modes are entered and exited. It also describes the behavior of TC1765 system components in all power management modes.

6.3.1 Idle Mode

Software requests the Idle Mode by setting the PMG_CSR.REQSLP bit field to 01_B.

After requesting Idle Mode, the Power Management State Machine (PMSM) posts an idle request signal to the CPU. The CPU finishes its current operation, sends an acknowledge signal back to the PMSM, and then enters an inactive state in which the CPU clocks and the DMU and PMU memory units are shut off.

In Idle Mode, memory accesses to the DMU and PMU via the FPI Bus cause these units to awaken automatically to handle the transactions. When memory transactions are complete, the DMU and PMU return to Idle Mode again.

The system will be returned to Run Mode through occurrence of any of the following conditions:

- An interrupt signal is received from an enabled interrupt source.
- An NMI request is received either from an external source via the $\overline{\text{NMI}}$ pin or from the Watchdog Timer. The Watchdog Timer triggers an NMI trap request in Idle mode when its count value (WDT_SR.TIM) transitions from $7FFF_H$ to 8000_H.
- An external power-on signal $\overline{\text{PORST}}$ or hardware reset signal $\overline{\text{HDRST}}$ is received.
- A software reset is requested by a FPI Bus agent by writing to the reset request register RST_REQ.

If any of these conditions arise, the TC1765 immediately awakens and returns to Run Mode. If it is awakened by a hardware or software reset signal, the TC1765 system begins its reset sequence. If it is awakened by a Watchdog Timer overflow event, it executes the instruction following the one which was last executed before Idle Mode was entered. If it is awakened by an NMI signal or interrupt signal, the CPU will immediately vector to the appropriate handler.

6.3.2 Sleep Mode

Software can request the Sleep Mode by setting PMG_CSR.REQSLP = 10_B.

6.3.2.1 Entering Sleep Mode

Sleep Mode is entered in two steps. In the first step, the CPU is put into Idle Mode in the same manner as described in **Section 6.3.1**. When the PMSM receives the Idle acknowledge signal back from the CPU, it goes on to the second step.

In the second step, a sleep signal is then broadcast on the FPI Bus. Each FPI Bus interface unit receives this signal. The response of each FPI Bus unit to the sleep signal

is determined by its own clock control register (CLC). These registers must have been previously configured by software.

6.3.2.2 TC1765 State During Sleep Mode

Sleep Mode is disabled for a unit if its CLC_EDIS bit field is 1. The sleep signal is ignored by this unit and it continues normal operation.

If the unit's clock control register bit CLC_EDIS is 0, Sleep Mode is enabled for this unit. In this case, the sleep signal will cause this unit to enter Sleep Mode. Two actions then occur:

1. The unit's bus interface finishes whatever transaction was in progress when the signal was received.
2. The unit's functions are suspended.

The TriCore architecture qualifies the actions in step 2 as follows. Depending on the module's Fast Shut-Off Enable bit CLC.FSOE in the clock control registers, the module's clocks are either immediately stopped (CLC.FSOE = 1), or the unit is allowed time to finish ongoing operations (CLC.FSOE = 0) before the clocks are stopped. For example, setting CLC.FSOE to 1 for a serial port will stop all actions in the serial port immediately when the sleep signal is received. Ongoing transmissions or receptions will be aborted. If CLC.FSOE is 0, ongoing transmissions or receptions will be completed, and then the clock will be shut off. The purpose of setting CLC.FSOE = 1 is to allow a debugger to observe the internal state of a peripheral unit immediately.

Please refer to the respective peripheral unit chapters for discussions of the exact implementation of Sleep Mode (Clock Control Register) for a specific peripheral unit.

6.3.2.3 Exiting Sleep Mode

The system will be returned to Run Mode by the same events that exit Idle Mode, as described in **Section 6.3.1**. The response of the CPU to being awakened is also the same as for Idle Mode. Peripheral units which have entered Sleep Mode will switch back to their selected Run Mode operation.

6.3.3 Deep Sleep Mode

In Deep Sleep Mode, the PMSM shuts off all clocks, the PLL, and the oscillator. Therefore, Deep Sleep Mode consumes the least power of all TC1765 states.

Deep Sleep Mode is requested through software by setting the PMG_CON.DSREQ bit to 1. The request bits for Deep Sleep Mode have been separated intentionally from the Idle Mode and Sleep Mode request bits to minimize the chance of inadvertently invoking Deep Sleep Mode.

Because no clock is running in a system in Deep Sleep Mode, it can not be awakened by any interrupt source, including the Watchdog Timer. It can only be awakened when it

receives an external reset or $\overline{\text{NMI}}$ signal, as described in this section. Software must prepare the external environment of the TC1765 to cause one of these signals under the appropriate conditions before entering Deep Sleep Mode. If Deep Sleep Mode were entered unintentionally without an event of this nature first being prepared, the TC1765 might never emerge from Deep Sleep Mode. For this reason, the PMG_CON register which sets up Deep Sleep Mode is specially protected. In order to write to PMG_CON, the WDT_CON0.ENDINIT bit must be set to 0 through a password-protected access mechanism to register WDT_CON. In order for the request to be activated, WDT_CON0.ENDINIT must first be set to 1 after the write to PMG_CON.

6.3.3.1 Entering Deep Sleep Mode

Deep Sleep Mode is entered in three steps. In the first step, the CPU is put into Idle Mode in the same way as described in **Section 6.3.1**. When the PMSM receives the Idle acknowledge signal back from the CPU, it goes on to the second step in which the PMSM activates the sleep signal, as described in **Section 6.3.2**. In the third step, the PMSM shuts off all clocks, the PLL, and the oscillator.

Note: The Power-On Reset Pin $\overline{\text{PORST}}$ should be kept stable when powering the TC1765 down.

Note: The software which turns on deep sleep mode must reside in the internal code scratch pad RAM to ensure that no external code accesses via the EBU are running when the PLL clock is shut down.

6.3.3.2 TC1765 State During Deep Sleep Mode

In Deep Sleep Mode, all port pins hold their state when Deep Sleep Mode is entered. The Deep Sleep Reset Enable Bit PMG_CON.DSRW controls whether the TC1765 is reset when Deep Sleep Mode is left.

- PMG_CON.DSRW = 0: TC1765 is not reset when Deep Sleep Mode is left.
- PMG_CON.DSRW = 1: TC1765 is reset when Deep Sleep Mode is left. Port pins are put into the reset state.

6.3.3.3 Exiting Deep Sleep Mode

Deep Sleep Mode can be exited in two ways:

- A power-on reset signal is detected ($\overline{\text{PORST}}$)
- The $\overline{\text{NMI}}$ pin detects a falling edge

When returning to full-power operation, the first step is to restart the oscillator and PLL, and re-enable the system clocks. This generally requires external hardware to wait until the PLL has had time to lock to its external clock source before the system can return to reliable operation.

Exactly how the TC1765 system returns from Deep Sleep Mode depends upon which signal re-awakens it. If awakened by a falling edge on the $\overline{\text{NMI}}$ pin, it further depends upon the state of the PMG_CON.DSRW bit.

6.3.3.4 Exiting Deep Sleep Mode With A Power-On Reset Signal

When awakened through a power-on reset signal ($\overline{\text{PORST}}$), the system initiates the same reset sequence as is used when power is first applied. The TC1765 automatically initiates its clock-acquisition sequence. This provides the time needed for the PLL to lock to the oscillator. The TC1765 will remain in the reset state until both the PLL is locked and the $\overline{\text{PORST}}$ signal is deactivated.

6.3.3.5 Exiting Deep Sleep Mode With an $\overline{\text{NMI}}$ Signal

The state of the Deep Sleep Reset Enable Bit, PMG_CON.DSRW, determines what happens when the TC1765 is awakened through a falling edge on the $\overline{\text{NMI}}$ pin.

If DSRW was set to 1 before entering Deep Sleep Mode, the TC1765 will execute a reset sequence similar to the power-on reset sequence. Therefore, all port pins are put into their reset state and stay in this state until they are affected in some way by boot operation or program execution. Note that the PLL configuration latched at the last power-on reset is still valid; the appropriate bits in register PLL_CLC have not been reset by entering and exiting Deep Sleep mode. Note also that the external NMI status flag NMISR.NMIEXT is not set after this Deep Sleep Mode exit case because the reset sequence clears NMISR.NMIEXT again after it has been set shortly through the falling edge on the $\overline{\text{NMI}}$ pin.

If DSRW was set to 0 before entering Deep Sleep Mode, a fast wake-up sequence is used. In this case, the TC1765 does not wait for the PLL to stabilize and lock to the external clock. Instead, it resumes operation as soon as the PLL provides clock signals. Note that the PLL configuration latched at the last power-on reset is still valid; the appropriate bits in register PLL_CLC have not been reset by entering and exiting Deep Sleep mode. The port pins continue to hold their state which was valid during Deep Sleep Mode until they are affected somehow by boot operation or program execution.

Special attention must be paid when using this type of wake-up. As soon as the device is woken up from Deep Sleep mode, the PLL begins generating clocks starting with the PLL's base frequency. When the external oscillator begins to generate clock signals, the PLL will begin to increase its frequency in order to achieve the programmed frequency ($f_{OSC} \times N$). Note that the start-up time of an external crystal oscillator can be in the range of some ms. This will continue until the PLL is locked to the external clock. Thus, since the TC1765 does not wait until the PLL has locked, its operation is based on a clock which will increase in frequency until the PLL is locked to the programmed frequency. Software can poll the PLL Lock Status bit (PLL_CLC.LOCK) for the lock status of the PLL.

Note: For wake-up through NMI, the NMI signal must held active until the clock system starts. Otherwise, the TC1765 will not enter the NMI trap handler routine.

6.3.4 Summary of TC1765 Power Management States

Table 6-3 summarizes the state of the various units of the TC1765 during Run Mode, Idle Mode, Sleep Mode, and Deep Sleep Mode.

Table 6-3 State of TC1765 Units During Power Management Modes

Unit	Run Mode	Idle Mode	Sleep Mode	Deep Sleep Mode
Oscillator & PLL	On	On	On	Off
CPU	Executing	Idle	Idle	Off (no clock)
DMU & PMU	Active	Idle, but accessible	Idle, but accessible	Off (no clock). Memory units hold their contents
Watchdog Timer	Functioning as programmed	Functioning as programmed	Functioning as programmed	Off (no clock)
FPI Bus Peripherals	Functioning as programmed	Functioning as programmed	Functioning as programmed	Off (no clock)
Debug Unit	Functioning	Functioning	Functioning	Off (no clock)
External Bus Controller (EBU)	Functioning as programmed	Functioning as programmed	Functioning as programmed	Off (no clock); The EBU pins hold the last value.
Ports	Functioning as programmed	Functioning as programmed	Functioning as programmed	Off (no clock). The EBU pins hold the last value.

7 Memory Map

The memory system of the TC1765 provides the following memories:

- Program Memory Unit (PMU) with
  - 8 KBytes Boot ROM (BROM)
  - 16 KBytes Code Scratch-Pad RAM (SPRAM)
  - 1 KByte Instruction Cache (ICACHE)
- Data Memory Unit (DMU) with
  - 32 KBytes Data Memory (SRAM)
  - Can be used for standby operation during power-down states using a separate power supply

This chapter gives an overview on the TC1765 memory map. Details on the specific features of the memories in the PMU and DMU modules are described in **Chapter 8** and **Chapter 9** of this User's Manual.

7.1 TC1765 Address Map

Table 7-1 defines the segment oriented address blocks of the TC1765 with its corresponding address range, size, and PMU/DMU access view.

Table 7-1 TC1765 Block Address Map

Segment	Address Range	Size	Description	DMU Acc.	PMU Acc.[1]	
0-7	$0000\ 0000_H$ - $7FFF\ FFFF_H$	2 GB	Reserved	–	–	
8	$8000\ 0000_H$ - $8FFF\ FFFF_H$	256 MB	Reserved	via FPI	PMU local	cached
9	$9000\ 0000_H$ - $9FFF\ FFFF_H$	256 MB	Reserved	DMU local	via FPI	
10	$A000\ 0000_H$ - $AFFF\ FFFF_H$	256 MB	External Memory Space	via FPI	via EBU or FPI	
11	$B000\ 0000_H$ - $BDFF\ FFFF_H$	224 MB	External Memory Space mappable into Segment 10	via FPI	via EBU or FPI	non-cached
	$BE00\ 0000_H$ - $BEFF\ FFFF_H$	16 MB	External Emulator Space		via FPI	
	$BF00\ 0000_H$ - $BFFF\ DFFF_H$	-16 MB	Reserved		PMU local	
	$BFFF\ E000_H$ - $BFFF\ FFFF_H$	8 KB	Boot ROM 4 KBytes general purpose 4 KBytes factory test support			
12	$C000\ 0000_H$ - $C000\ 3FFF_H$	16 KB	Local Code Scratch-Pad RAM (SPRAM)	via FPI	PMU local	
	$C000\ 4000_H$ - $C7FF\ FEFF_H$	–	Reserved			
	$C7FF\ FF00_H$ - $C7FF\ FFFF_H$	256 B	PMU Control Registers			
	$C800\ 0000_H$ - $CFFF\ FFFF_H$	–	Reserved			

Table 7-1 TC1765 Block Address Map (cont'd)

Segment	Address Range	Size	Description	DMU Acc.	PMU Acc.[1]	
13	D000 0000$_H$ - D000 7FFF$_H$	32 KB	Local Data Memory (SRAM)	DMU local	via FPI	non-cached
	D000 8000$_H$ - D7FF FEFF$_H$	–	Reserved			
	D7FF FF00$_H$ - D7FF FFFF$_H$	256 B	DMU Registers			
	D800 0000$_H$ - DFFF FFFF$_H$	256 MB	Reserved			
14	E000 0000$_H$ - EFFF FFFF$_H$	256 MB	External Peripheral and Data Memory Space	via FPI	not possible	
15	F000 0000$_H$ - F000 3BFF$_H$	-16 KB	On-Chip Peripherals & Ports	via FPI	not possible	non-cached
	F000 3C00$_H$ - F000 3DFF$_H$	512 B	DMA Registers			
	F000 3E00$_H$ - F00F FFFF$_H$	–	Reserved			
	F010 0000$_H$ - F010 0BFF$_H$	12 × 256 B	CAN Module			
	F010 0C00$_H$ - FFFE FEFF$_H$	–	Reserved			
	FFFE FF00$_H$ - FFFE FFFF$_H$	256 B	CPU Slave Interface Registers (CPS)			
	FFFF 0000$_H$ - FFFF FFFF$_H$	64 KB	Core SFRs + GPRs			

[1] The PMU can access external instruction memory directly ("via EBU") or via the FPI Bus ("via FPI"). If both paths are possible as defined in this column, selection is done via SCU_CON.EXTIF.

Segments 0-7

This memory range is a reserved area in the TC1765.

Segment 8

This memory segment is reserved in the TC1765. It is assigned to cached access purposes in future product derivatives.

Segment 9

This memory segment is reserved in the TC1765. It is assigned to cached access purposes in future product derivatives.

Segment 10

This 256 MBytes memory segment is assigned for external burst mode Flash code memories when operating in cached mode.

Segment 11

This memory segment contains a 224 MBytes reserved area for external code/data memory or external peripherals. All burst mode instruction fetches of the PMU to a code memory located in this area are non-cached. A code memory in Segment 11 can be mapped into Segment 10. Segment 11 also contains a 16 MBytes external area reserved for the emulator and the 8 KBytes Boot ROM (BROM).

External data memory and external peripherals located in the address region B000 0000_H to BEFF $FFFF_H$ are accessible via EBU and FPI Bus.

External code memory located in the address region B000 0000_H to BDFF $FFFF_H$ is only accessible using chip select signal $\overline{CS0}$.

External emulator memory located in the address region BE00 0000_H to BEFF $FFFF_H$ is only accessible via the EBU and FPI Bus using any chip select signal.

Segment 12

This memory segment contains the 16 KBytes Local Code Scratch-Pad RAM (SPRAM) operating in non-cached mode. It also includes the control registers of the PMU.

Segment 13

This memory segment contains the 32 KBytes Local Data Memory (SRAM) and the DMU control registers.

Segment 14

This memory segment is a non-cached 256 MBytes segment reserved for external peripherals and external data memory.

Segment 15

This memory segment is dedicated for CPU, on-chip peripheral units, and ports (see **Table 7-2**).

*Note: Accesses to address regions defined as “Reserved” in **Table 7-1** lead to a bus error or a CPU trap depending on data or instruction fetch access.*

7.2 Memory Segment 15 - Peripheral Units

Table 7-2 shows the block address map of Segment 15.

Table 7-2 Block Address Map of Segment 15

Symbol	Description	Address Range	Size
SCU	System Control Unit	$F000\ 0000_H$ - $F000\ 00FF_H$	256 Bytes
–	Reserved	$F000\ 0100_H$ - $F000\ 01FF_H$	–
BCU	Bus Control Unit	$F000\ 0200_H$ - $F000\ 02FF_H$	256 Bytes
STM	System Timer	$F000\ 0300_H$ - $F000\ 03FF_H$	256 Bytes
OCDS	On-Chip Debug Support	$F000\ 0400_H$ - $F000\ 04FF_H$	256 Bytes
EBU	External Bus Unit	$F000\ 0500_H$ - $F000\ 05FF_H$	256 Bytes
–	Reserved	$F000\ 0600_H$ - $F000\ 06FF_H$	–
GPTU	General Purpose Timer Unit	$F000\ 0700_H$ - $F000\ 07FF_H$	256 Bytes
ASC0	Async./Sync. Serial Interface 0	$F000\ 0800_H$ - $F000\ 08FF_H$	256 Bytes
ASC1	Async./Sync. Serial Interface 1	$F000\ 0900_H$ - $F000\ 09FF_H$	256 Bytes
SSC0	High-Speed Synchronous Serial Interface 0	$F000\ 0A00_H$ - $F000\text{-}0AFF_H$	256 Bytes
SSC1	High-Speed Synchronous Serial Interface 1	$F000\ 0B00_H$ - $F000\text{-}0BFF_H$	256 Bytes
–	Reserved	$F000\ 0C00_H$ - $F000\ 17FF_H$	–
GPTA	General Purpose Timer Array	$F000\ 1800_H$ - $F000\ 1FFF_H$	8×256 Bytes
–	Reserved	$F000\ 2000_H$ - $F000\ 21FF_H$	–
ADC0	Analog-to-Digital Converter 0	$F000\ 2200_H$ - $F000\ 23FF_H$	2×256 Bytes
ADC1	Analog-to-Digital Converter 1	$F000\ 2400_H$ - $F000\ 25FF_H$	2×256 Bytes
–	Reserved	$F000\ 2600_H$ - $F000\ 27FF_H$	–
P0	Port 0	$F000\ 2800_H$ - $F000\ 28FF_H$	256 Bytes
P1	Port 1	$F000\ 2900_H$ - $F000\ 29FF_H$	256 Bytes
P2	Port 2	$F000\ 2A00_H$ - $F000\ 2AFF_H$	256 Bytes
P3	Port 3	$F000\ 2B00_H$ - $F000\ 2BFF_H$	256 Bytes
P4	Port 4	$F000\ 2C00_H$ - $F000\ 2CFF_H$	256 Bytes
P5	Port 5	$F000\ 2D00_H$ - $F000\ 2DFF_H$	256 Bytes

Table 7-2 Block Address Map of Segment 15 (cont'd)

Symbol	Description	Address Range	Size
–	Reserved	F000 2E00$_H$ - F000 3BFF$_H$	–
DMA	DMA Registers	F000 3C00$_H$ - F000 3DFF$_H$	2 × 256 Bytes
–	Reserved	F000 3E00$_H$ - F00F FFFF$_H$	–
CAN[1)]	Controller Area Network Module	F010 0000$_H$ - F010 0BFF$_H$	12 × 256 Bytes
–	Reserved	F010 0C00$_H$ - FFFE FEFF$_H$	–
CPU	Slave Interface Registers (CPS)	FFFE FF00$_H$ - FFFE FFFF$_H$	256 Bytes
	Reserved	FFFF 0000$_H$ - FFFF BFFF$_H$	–
	Memory Protection Registers	FFFF C000$_H$ - FFFF EFFF$_H$	12 KBytes
	Reserved	FFFF F000$_H$ - FFFF FCFF$_H$	–
	Core Debug Register (OCDS)	FFFF FD00$_H$ - FFFF FDFF$_H$	256 Bytes
	Core Special Function Registers (CSFRs)	FFFF FE00$_H$ - FFFF FEFF$_H$	256 Bytes
	General Purpose Register (GPRs)	FFFF FF00$_H$ - FFFF FFFF$_H$	256 Bytes

1) Access to unused address regions within this peripheral unit don't generate a bus error.

*Note: Accesses to all reserved address regions within a peripheral unit or an address block normally lead to a bus error. Detailed definitions on access modes and bus errors are given in **Chapter 20** in this User's Manual.*

8 Program Memory Unit

The Program Memory Unit PMU controls the CPU code fetches from internal and external code memory. The PMU consists of the functional blocks (**Figure 8-1**):

- 8 KByte Boot ROM memory (BROM)
- 16 KByte scratch-pad code RAM (SPRAM)
- 1 KByte instruction cache (ICACHE)
- PMU control block
- Interface to the CPU Instruction Fetch Unit
- Interface to the EBU for external code fetches
- FPI Bus interface

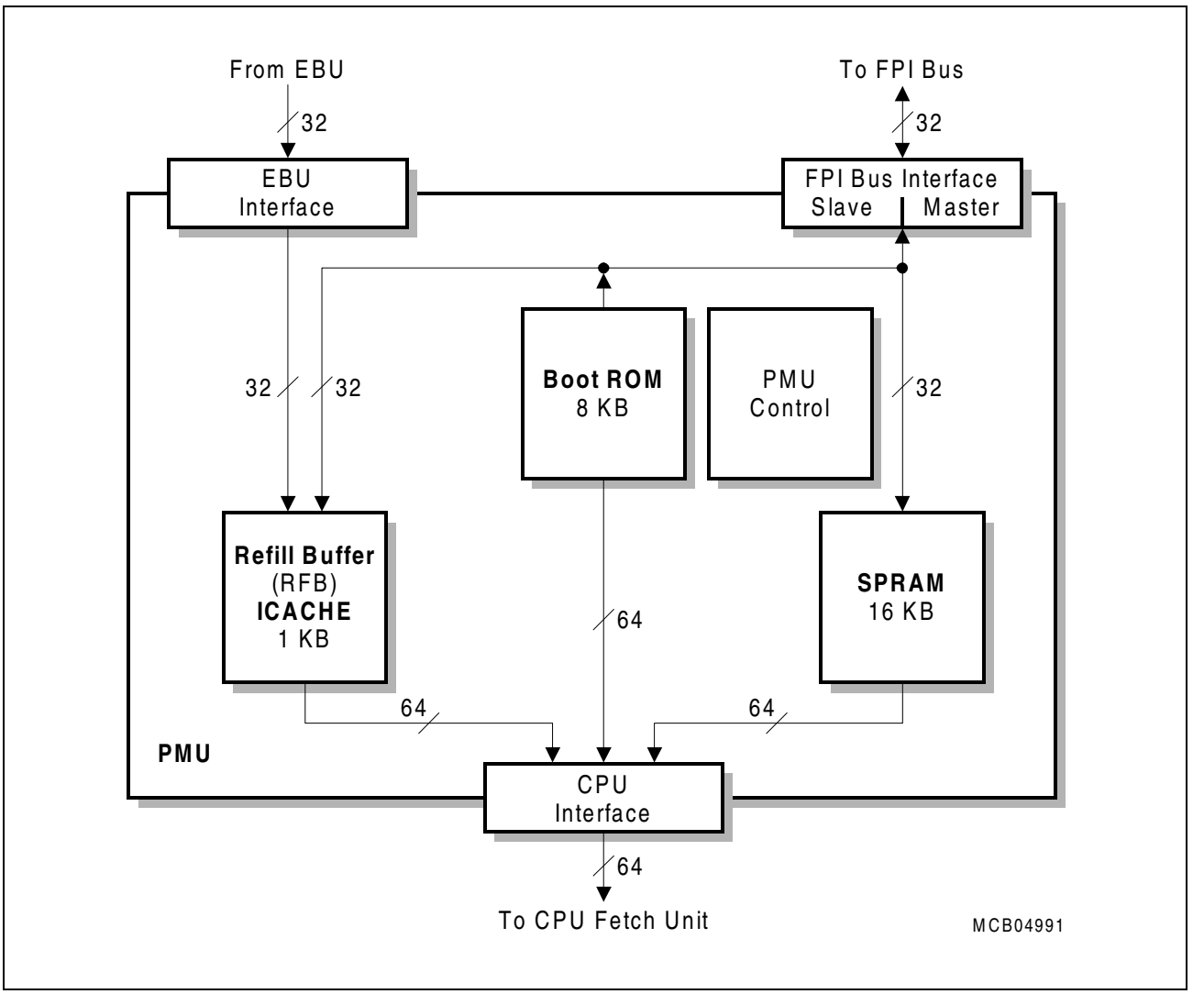


Figure 8-1 PMU Block Diagram with Data Paths

The FPI Bus interface is a master/slave interface which handles all transactions between FPI Bus and the PMU code memories. The master part of the interface is used when the PMU needs to access resources which are located on the FPI Bus. The slave part of the

interface is used when another FPI Bus master needs to access PMU resources such as the CSRAM.

The EBU interface is used for external instruction fetches from external burst mode Flash memory devices.

The Instruction Cache contains the cache RAM with the tag RAM. The Refill Buffer is mainly required for instruction code assembling and alignment as well as for external Burst Flash access synchronization with the internal clock of the PMU. It can be accessed while a cache refill is performed “Hit under Refill”.

8.1 Memories Controlled by PMU

Table 8-1 summarizes the sections of TC1765 internal and external code/program memories that are controlled by the PMU.

Table 8-1 Address Map of PMU Related Memories

Segment	Address	Name	Description
On-Chip Memory			
11	BFFF E000$_H$ - BFFF FFFF$_H$	BROM	Boot ROM non-cached
12	C000 0000$_H$ - C000 3FFF$_H$	SPRAM	Local Scratch-Pad Code RAM; non-cached
External Program Memory			
10	A000 0000$_H$ - AFFF FFFF$_H$	ExtMem	External Program Memory, cached
11	B000 0000$_H$ - BDFF FFFF$_H$		External Program Memory, mappable into segment 10; non-cached

8.2 Scratch-Pad RAM, SPRAM

The Scratch-Pad RAM (SPRAM) is a 16 KByte static RAM. As a code memory, it is assigned especially to hold code that must be executed very fast (e.g. interrupt routines).

The SPRAM can be accessed from the FPI Bus side by another bus master, such as the Data Memory Unit, DMU. On a read access from the FPI Bus (possible in supervisor mode as well as in user mode), the data width can be only 32 bits (word) wide. The natural alignment of the accessed data must be obeyed, that is, bytes can be aligned on any byte boundary, halfwords must be aligned to halfword (even byte) boundaries, and word accesses must be aligned to word boundaries. Accesses not following this rule will be flagged with an FPI Bus error by the PMU.

On a write access from the FPI Bus (only possible in supervisor mode!), the data width can only be 32 bits wide and must be aligned to word boundaries. Byte and halfword accesses are not allowed.

CPU fetch accesses to the address range of the SPRAM are never cached in the ICACHE. They are always directly targeted to the SPRAM. A code fetch access from the CPU to the SPRAM can be performed in one clock cycle, the data width of such an access is 64 bits. Note that the CPU Fetch Unit can only read from the SPRAM and never write to it.

8.3 Instruction Cache, ICACHE

The ICACHE of the PMU is a one-way set-associative cache with a size of 1 KByte. It is organized in 32 cache lines with 32 bytes each. A 34-bit tag information field is assigned to each cache line (see **Figure 8-2**).

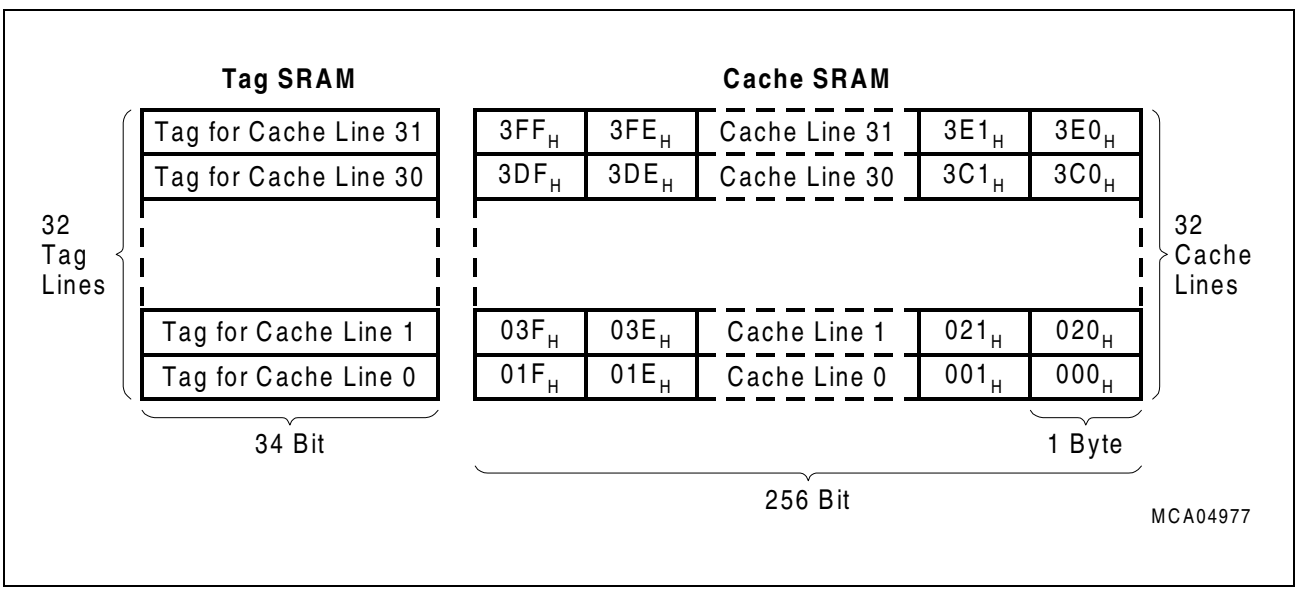


Figure 8-2 CACHE Organization

8.3.1 Cache Organization

The organization of the ICACHE is 32 cache lines with 32 bytes per line. Each cache line is divided into eight words (32 bits) with a valid bit in the tag line for each word. Alignment of a cache line results to an 8-word address line border (address bits A[4:0] = 0). With the 32/16-bit mixed instruction set formats of the TriCore, a full cache line can hold a minimum of eight 32-bit instructions and a maximum of sixteen 16-bit instructions.

The address of a CPU instruction fetch is first decoded to determine the access target (for example, Scratch Pad RAM, External Flash, address range accessible via FPI Bus, cachable area). All CPU instruction fetch accesses in the address ranges of the cachable area (segments 10) are targeted to the Refill Buffer. If the ICACHE is enabled and ICACHE bypass disabled, the ICACHE is also targeted. If the address and its associated instruction are found in the cache (Cache Hit), the instruction is passed to the CPU's Fetch Unit. If the address is not found in the cache (Cache Miss), the PMU's cache controller issues a cache refill sequence.

8.3.2 Cache Bypass Control

The ICACHE can be bypassed as controlled by bit PMU_CON.CBP. The default value for bit CBP after reset is 1, thus bypassing of the ICACHE is enabled. If CBP is 0 (bypass disabled), bit PMU_CON.CDIS must be cleared for enabling the ICACHE.

Table 8-2 ICACHE Enable and Bypass Control

CDIS	CPB	Description
0	0	Refill Buffer and ICACHE accessible
0	1	Refill Buffer accessible; ICACHE not accessible but cache lines remain valid
1	0	Refill Buffer accessible; ICACHE not accessible and cache lines are set invalid (equal to an ICACHE flush operation)
1	1	

8.3.3 Refill Buffer

The Refill Buffer can be assumed to be a small cache, 256 bits wide and divided into eight words (32-bits) with a valid bit for each word. It is mainly required for instruction code assembling and alignment as well as for external Burst Flash access synchronization with the internal clock of the PMU. The Refill Buffer can be accessed while a cache refill is performed "Hit under Refill".

The following data sources are handled by the Refill Buffer:

- FPI Bus data (32-bit wide)
- Instruction fetch bus (external access data, 32-bit wide)

8.3.4 Refill Sequence for Cache and Refill Buffer

Cache refill is performed with a Critical Word First strategy. This means that the refill sequence starts with the instruction actually requested (the critical word) by the CPU Fetch Unit and continues to the end of the cache line. A refill will always be done in 32-bit quantities. If the critical word maps onto the first 32-bit entry in the cache line, a refill of the entire cache line, eight words, will be performed. If the critical word maps onto the last 32-bit entry of a cache line, only this word will be refilled. In any case all valid bits of the refilled cache line are cleared. Thus, depending on the location of the critical word, the refill sequence will always be from one up to eight words without wrap-around (the instructions mapping to the refilled cache line which are on addresses lower than that of the critical word are not fetched, except for instructions located within the word containing the critical word). A refill sequence will always only affect one cache line. There is no prefetching of the next cache line (no crossing of lines).

8.3.5 Cache Flush Operation

The ICACHE and the Refill Buffer are flushed (cache lines are set as invalid) when bit PMU_CON.FLACC is set. This bit remains set as long as the ICACHE and RFB flush operation is ongoing and is reset when it is finished.

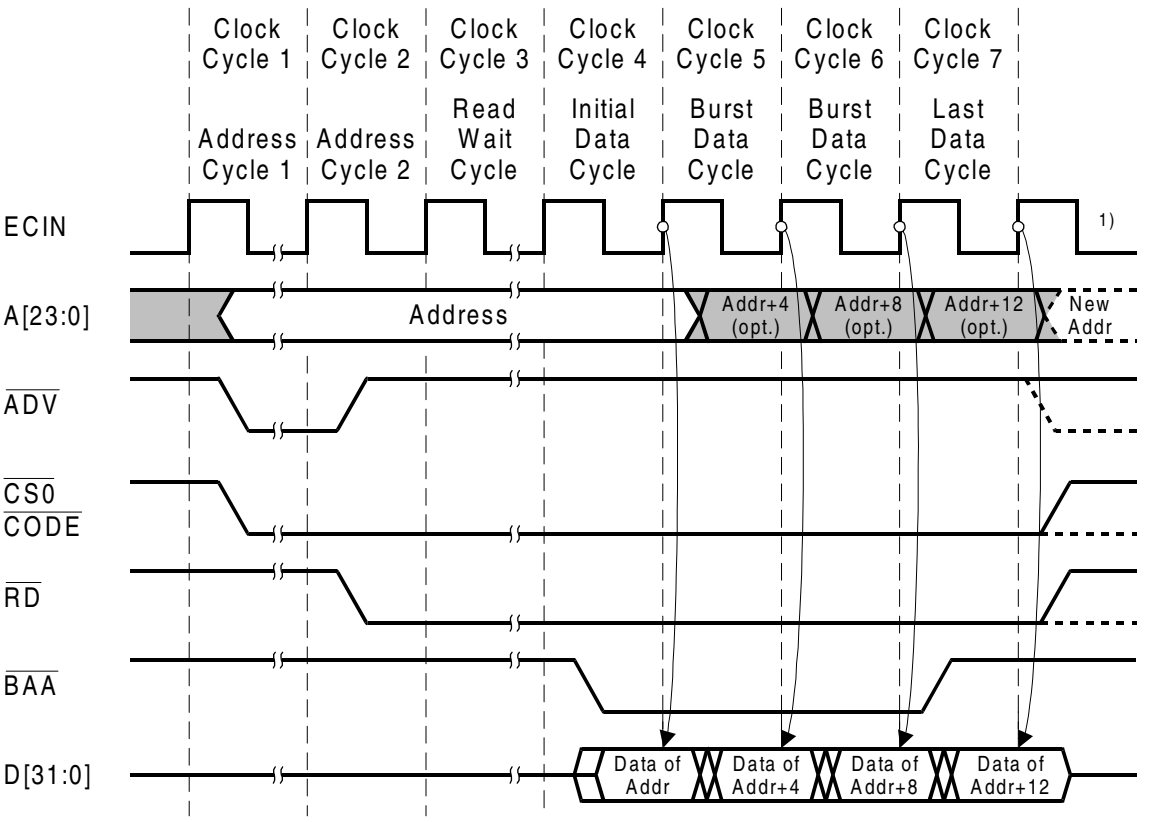
The Refill Buffer is flushed (cache lines are set as invalid) when bit PMU_CON.FLRFB is set. This bit remains set as long as the RFB flush operation is ongoing and is reset when it is finished.

8.4 External Code Fetches via External Bus Interface Unit

The PMU interface to the EBU is especially designed to perform burst mode cycle operations to an external burst flash code memory. Burst mode code fetch operations requested by the PMU use the EBU with its interface signals. During these burst mode code fetch operations, external accesses via the FPI Bus Interface of the EBU are delayed.

- 16-bit/32-bit data bus width
- The timings of PMU burst mode cycle operations are controlled by the PMU External Instruction Fetch Control Register (see **Section 8.6.2**). The following parameters of burst mode cycle operations can be selected:

Figure 8-3 shows a basic timing of synchronous burst mode operation. Other examples on burst mode timings are included in **Chapter 13** of this User's Manual.



1) The dotted waveforms indicate the start of a new address cycle.

Example settings: Burst of 4 words, ADVLEN = 0, RDWLEN = 1, OIDA2 = 0
Burst data address output optional, enabled via bit DAEN.

MCT04999

Figure 8-3 Synchronous Burst Read Operation Example

8.5 Boot ROM

The TC1765 contains 8 KByte of Boot ROM memory, which can be used for:

- Device operating mode initialization routines
- Bootstrap loader support
- Test functions

8.5.1 Bootstrap Loader Support

An integrated bootstrap mechanism is provided to support a system start with boot operation after reset. If the boot mode is selected during reset, program execution is started out of the Boot ROM. The functionality of the boot routine will be similar to the one which was implemented in other 16-Bit Infineon microcontrollers.

Note: The bootstrap loader and the functionality of the boot routines will be described in detail in a special document.

8.6 PMU Registers

As shown in **Figure 8-4**, the following control register are implemented in the PMU. These registers and their bits are described in this section.

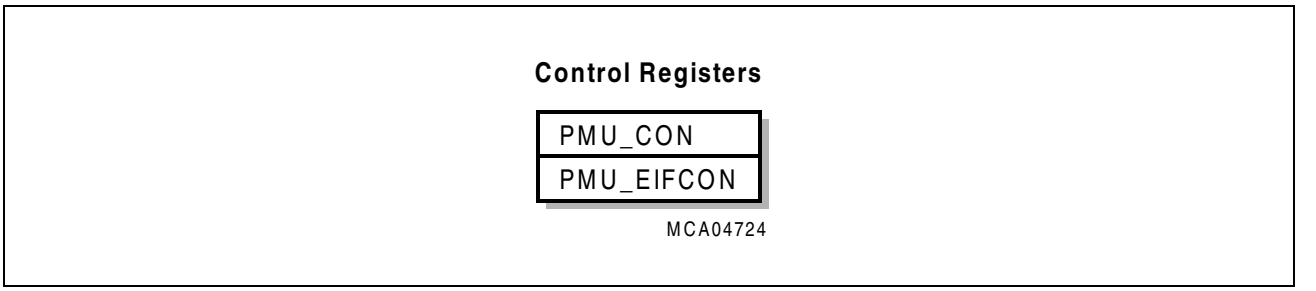


Figure 8-4 PMU Registers

Table 8-3 PMU Registers

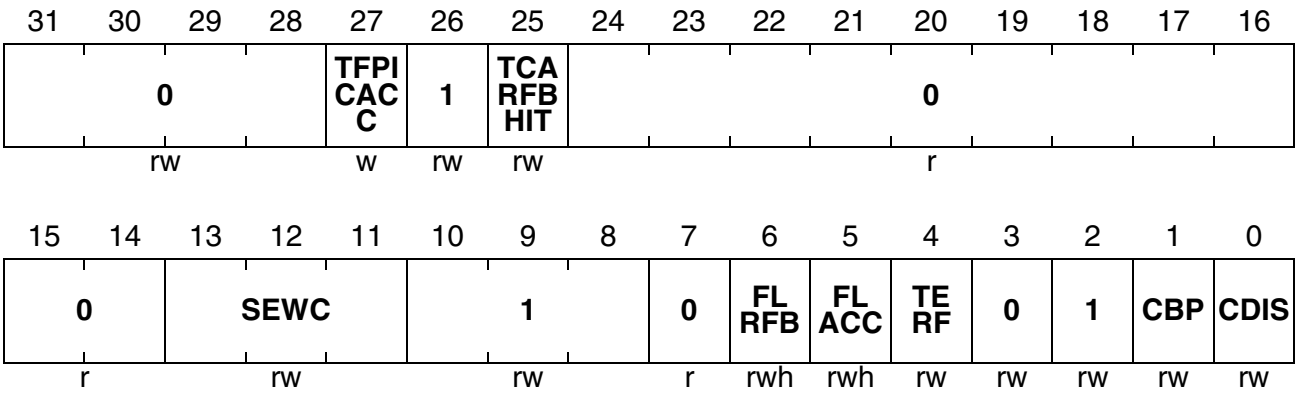
Register Short Name	Register Long Name	Offset Address	Description see
PMU_CON	PMU Control Register	0010_H	**Page 8-8**
PMU_EIFCON	PMU External Instruction Fetch Control Register	0018_H	**Page 8-10**

In the TC1765, the registers of the PMU are located in the following address range:

- Module Base Address: $C7FF\ FF00_H$
  Module End Address: $C7FF\ FFFF_H$
- Absolute Register Address = Module Base Address + Offset Address
  (offset addresses see **Table 8-3**)

8.6.1 PMU Control Register

PMU_CON
PMU Control Register **Reset Value: 0400 3F06$_H$**

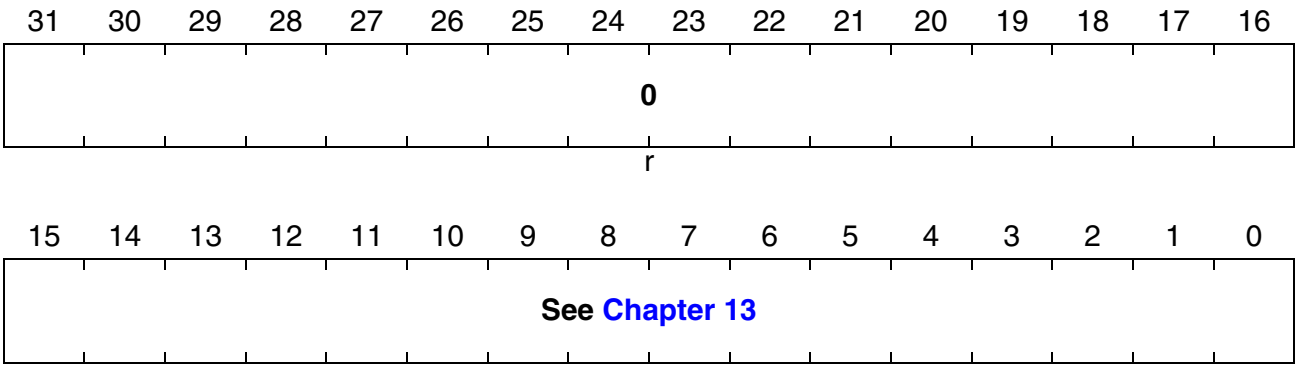


Field	Bits	Type	Description
CDIS	0	rw	**Instruction Cache Disable** 0 ICACHE enabled (default after reset) 1 ICACHE disabled
CBP	1	rw	**Cache Bypass** 0 ICACHE is not bypassed 1 ICACHE is bypassed (default after reset)
1	2	rw	**Reserved** Bit is 1 after reset and can be read/written without any function. A 0 is read if a 0 written before.
0	3	rw	**Reserved** Bit is 0 after reset and can be read/written without any function. A 1 is read if a 1 written before.
TERF	4	rw	**Terminate External Refill on a New Miss** 0 RFB refill operation is not terminated on a cache miss which occurs on an access to an external burst mode code memory (default after reset). 1 RFB refill operation is terminated on a cache miss which occurs on an access to an external burst mode code memory.
FLACC	5	rwh	**Flush Caches** 0 No flush operation (default after reset) 1 An ICACHE and Refill Buffer flush operation is executed Bit is reset by hardware after the ICACHE flush operation has been finished.

Field	Bits	Type	Description
FLRFB	6	rwh	**Flush Refill Buffer** 0 No flush operation (default after reset) 1 Refill Buffer flush operation is executed Bit is reset by hardware after the Refill Buffer flush operation has been finished.
1	[10:8]	rw	**Reserved** Bit is 1 after reset and can be read/written without any function. A 0 is read if a 0 written before.
SEWC	[13:11]	rw	**Subsequent Access Wait Cycles for External Flash** Controls minimum wait cycles between consecutive burst accesses from external Flash. 0_H-7_H 0 to 7 wait cycles inserted (default after reset are 7 cycles)
TCARFBHIT	25	rw	**Test Cache/Refill Buffer Hit** This bit should be always written by the user with 0.
1	26	rw	**Reserved** Bit is 1 after reset and should be written with 1.
TFPICACC	27	w	**Test for FPI Cache Access** 0 Normal operation. 1 Test mode with SPRAM switched off and cache visible is enabled; Bit is always read as 0. This bit should be always written with 0 by the user.
0	[31:28]	rw	**Reserved** Bit is 0 after reset and should be written with 0.
0	7, [24:14]	r	**Reserved**; read as 0; should be written with 0.

8.6.2 External Instruction Fetch Control Register

PMU_EIFCON
PMU External Instruction Fetch Control Register **Reset Value: see Chapter 13**



This register defines the operation of the burst mode instruction fetches.
The detailed description can be found in **Chapter 13**.

9 Data Memory Unit

The Data Memory Unit (DMU) shown in **Figure 9-1**, contains:

- 32 KByte data memory (SRAM)
  – Can be used for standby operation (separate power supply pin $V_{DDSBRAM}$)
- DMU control block
- Interface to the CPU Load/Store Unit
- Interface to the FPI Bus

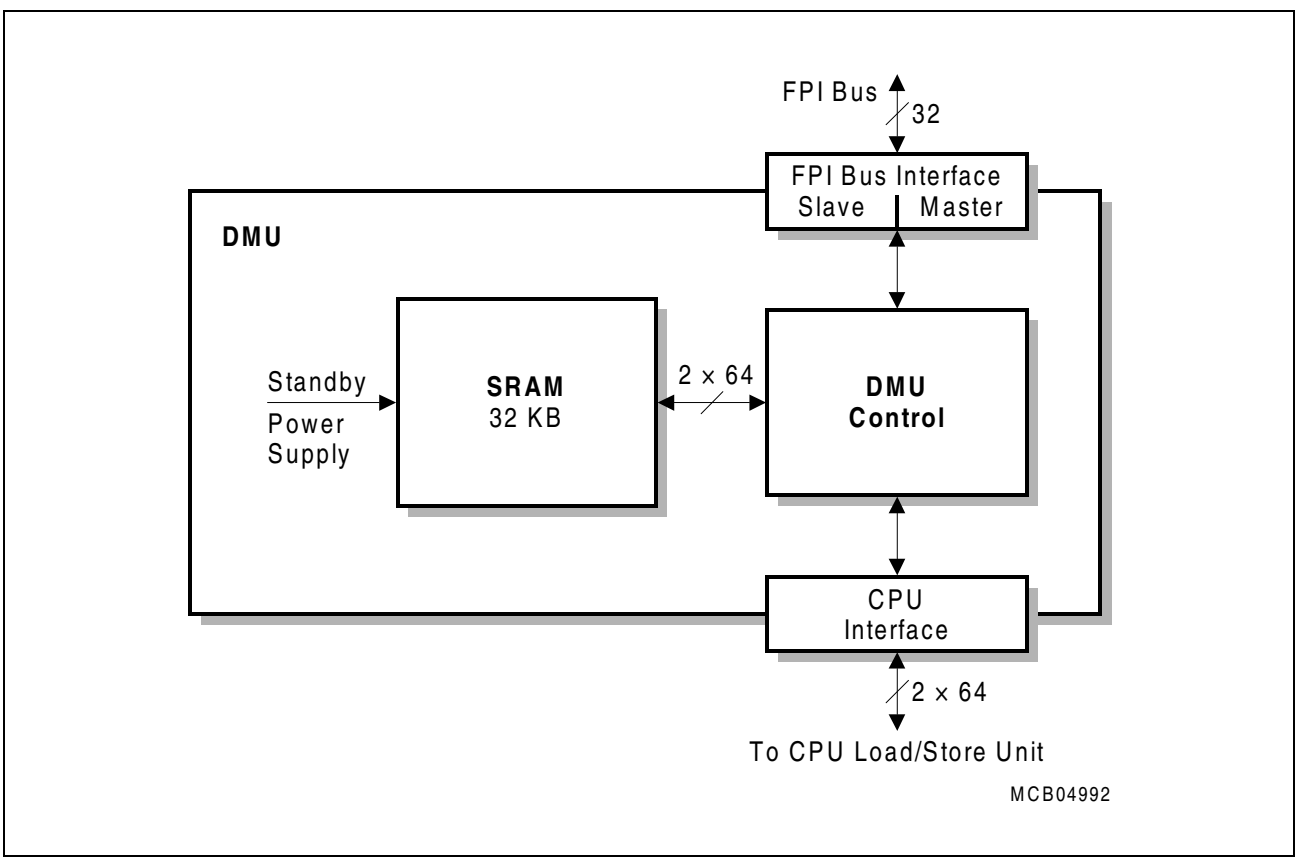


Figure 9-1 Block Diagram of the Data Memory Unit (DMU)

The FPI Bus interface of the DMU can operate in either master or slave mode. The master part of the interface is used when the CPU Load/Store Unit requests a data access to a data resource that is outside the DMU on the FPI Bus (for example, a module connected to the FPI Bus, such as the External Bus Control Unit (EBU)). The slave part of the interface is required when another FPI Bus master (such as the DMA controller) needs to access the DMU data memory.

The data width for read and write accesses to/from the data memory within the DMU via the FPI Bus can be 16- or 32-bit (byte, half-word or word). The natural alignment of the accessed data must be obeyed, that is, half-words must be aligned to half-word (even byte) boundaries, and word accesses must be aligned to word boundaries. Accesses not following this rule will be flagged with a bus error by the DMU.

The data memory is located at the beginning of the non-cacheable Segment 13.

Table 9-1 DMU Address Map

Segment	Address	Name	Description	CPU Access		FPI Bus Access	
				Load	Store	Read	Write
13	$D000\ 0000_H$ - $D000\ 7FFF_H$	SRAM	Data Memory	1)		2)	
	$D000\ 8000_H$ - $D7FF\ FEFF_H$	Reserved		3)		BE	BE
	$D7FF\ FF00_H$ - $D7FF\ FFFF_H$	–	DMU Registers	1)		2)	
	$D800\ 0000_H$ - $DFFF\ FFFF_H$	Reserved		3)		BE	BE

1) CPU Load/Store accesses to this range can be performed in User or Supervisor Mode. Access width can be 8-, 16-, 32-, or 64-bit, with 8-bit data aligned on byte boundaries and all others aligned on half-word (16-bit) boundaries. Misaligned accesses to the data memory by the CPU's Load/Store Unit will not occur since such conditions will already be handled inside the CPU (Unalignment trap, ALN).

2) The read/write accesses from the FPI Bus can be performed in User or Supervisor Mode. Access width can be 16- or 32-bit, with data aligned on its natural boundary. Misaligned access will result in a bus error.

3) This range is reserved and load/store accesses will be flagged with a Load/Store Range Error Trap.

The placement of the SRAM into the lower half of Segment 13 facilitates the use of the absolute addressing mode for load and store operations, supporting fast access to data stored in the lower 16 KBytes of the data memory (in absolute addressing mode, an address in the lower 16 KBytes of each of the segments can be specified as an immediate address of a load/store instruction; such an address does not have to first be loaded into an address register).

Note: Read-modify-write instructions from FPI Bus to DMU memory are locked.

9.1 DMU Trap Generation

Several error conditions can lead to a trap being reported by the DMU back to the CPU. These include range errors, DMU control register access errors, and FPI Bus errors.

To facilitate a detailed analysis of an error/trap, the DMU provides two read-only status registers that hold information about the type of the error. The Synchronous Trap Flag Register (DMU_STR) contains the flags indicating the cause of a synchronous trap, while the Asynchronous Trap Flag Register (DMU_ATR) holds the flags for the cause of an asynchronous trap. In the TC1765, load operations are always synchronous, while store operations are asynchronous to the instruction stream.

In general, whether an operation in the DMU can result in a synchronous or asynchronous error trap depends on the actual condition and sequence of operation in the DMU. Thus, for each of the possible DMU error scenarios, an error flag is provided in both registers DMU_STR and DMU_ATR. When an error is detected in the DMU, the respective trap signal is generated to the CPU and the appropriate bit in the associated trap flag registers is set.

The Trap Service Routine (TSR) invoked through the trap then needs to read the appropriate DMU Trap Flag Register to further determine the root cause of the trap. Reading a DMU Trap Flag Register in Supervisor Mode returns the contents of the register, and then clears the register to 0. Reading a trap flag register in User Mode only returns the contents of the register, but leaves it unaltered. The latter operation is implemented to allow debuggers/emulators to examine the status of the trap flag register without modifying it. The TSRs of user application code should always read these registers in Supervisor Mode in order to clear their contents.

9.1.1 FPI Bus Error

Two status flags are implemented to indicate an FPI Bus error. One flag indicates errors resulting from a FPI Bus store operation, the other one indicates errors resulting from a FPI Bus load operation. The appropriate flags (DMU_STR.LFESTF or DMU_STR. SFESTF) are set if a DMU operation to or from the FPI Bus is performed, and an error occurs on the FPI Bus.

Please note that in case of FPI Bus errors, an FPI Bus error interrupt is generated by the Bus Control Unit (BCU) separate from the DMU trap generation. The appropriate trap service routine in the application code needs to take this into account and should also handle the interrupt request from the BCU.

9.1.2 Range Error

Range errors are caused by accesses to reserved address ranges in the DMU. Accesses to address ranges in Segment 13 (DMU) which are not covered by the data memory or the DMU control register ranges will lead to a range trap.

In the TC1765, the entire Segment 9 is reserved, no memory is implemented in this range. Accesses to Segment 9 will always cause a DMU Range Error Trap.

In each of the DMU trap flag registers, two status flags are implemented to indicate a range error. One flag indicates errors resulting from a store operation (DMU_ATR. SRESTF), the other one indicates errors resulting from a load operation (DMU_STR. LRESTF). The appropriate flag is set if an access to the reserved address ranges is performed.

9.1.3 DMU Register Access Error

DMU register access errors are caused if an improper access to a DMU register is performed.

CPU load/store access to the DMU registers must only be made with double-word-aligned word accesses. An access not conforming to this rule, or an access that does not follow the specified privilege mode (supervisor mode, EndInit-protection), or a write access to a read-only register, will lead to a DMU Control Register Error trap. An access to reserved locations within the DMU register address area will not be flagged with an error. A read will return all zeros, a write will have no effect.

In each of the DMU trap flag registers, two status flags are implemented to indicate a register access error. One flag indicates errors resulting from a store operation (DMU_ATR.SCESTF), the other one indicates errors resulting from a load operation (DMU_STR.LCESTF).The appropriate flag is set if an improper access to the DMU registers is performed.

9.1.4 Cache Management Error

Cache management errors are generated when one of the special cache instructions, DFLUSH, DINV and DFLINV, specify a non-cacheable address.

Note: Because of a missing data cache, these instructions should not be used in the TC1765.

9.2 Overlay Functionality

Overlay functionality provides the means to redirect data fetches from the code memory to dedicated areas within the internal and external data memory range. The purpose of this functionality is to reprogram parameters stored normally in the code memory during run time. The instruction fetch is not affected by redirection. **Table 9-2** shows the access and redirection types.

Table 9-2 Access Types and Redirection Types

Access Type	Redirection	
	to internal Data Memory (Segment D)	**to external Data Memory (Segment A, B, E)**
Data Access from external Code Memory[1)]	**Section 9.2.1**	**Section 9.2.2**
Data Access from external Code Memory (using Signal $\overline{\text{CODE}}$ and $\overline{\text{CS0}}$)	not possible	**Section 9.2.3**

1) Note that byte, half-word, and word data accesses to the Code Memory can be redirected. Double-word accesses are always directed to the Code Memory.

The following sections describe the initial access type (from …) and the type of redirection (… to …).

9.2.1 Redirection From External Code to Internal Data Memory

Data reads from external code memory (Segment 10 or 11) can be redirected to internal data memory (Segment 13), as shown **Figure 9-2**.

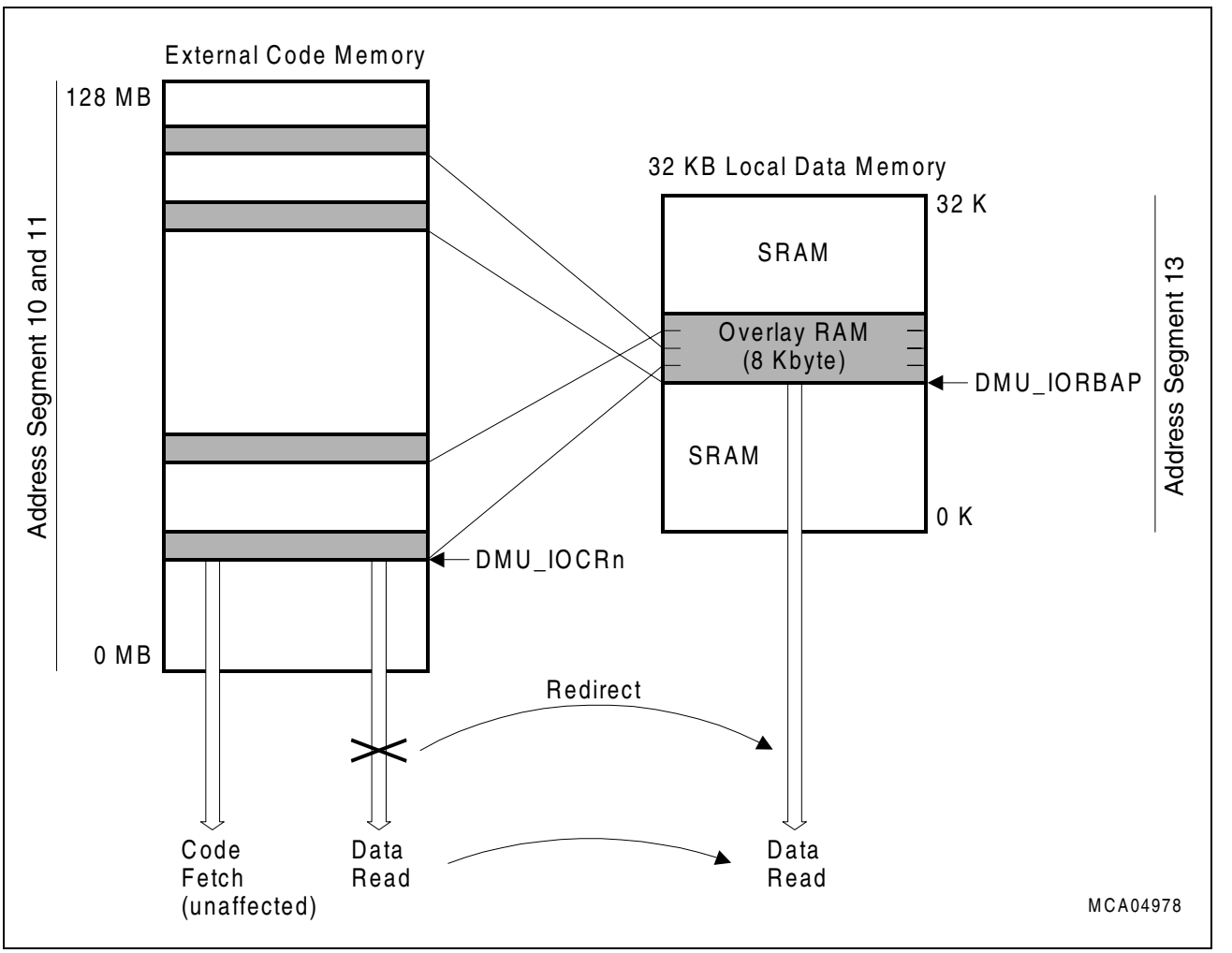


Figure 9-2 Redirection From External Code to Internal Data Memory

The total size of the internal Overlay RAM area is 8 KBytes divided into four blocks of 2 KBytes each. The Overlay RAM area is located on 8-KByte page boundaries within the internal data memory (Segment 13). The start address of the Overlay RAM area is specified in the Internal Overlay RAM Base Address Page Register (DMU_IORBAP). Each of the four 2-KByte blocks within the Overlay RAM can be individually enabled for overlay functionality by bit OEN in the corresponding Internal Overlay Control Register DMU_IOCRn (n = 3-0).

There are four overlay RAM control registers (DMU_IOCRn, n = 3-0) assigned to control the internal overlay functionality. Each register specifies the start address of an overlayed 2-KByte block within the lower 128 MBytes of Segment 10 and 11. This start address can be placed on any 2-KByte boundary within the external code memory, using bit field OVPTR.

The resulting address on a data read to the external code memory is compared with the addresses stored in the Internal Overlay Control Register. If the overlay function is enabled and an address match occurs, an read access to the corresponding address in the Overlay RAM area is performed. The access to the external code memory is performed in parallel and determines the timing of the redirected data access.

9.2.2 Redirection From External Code to External Data Memory

Data reads from external code memory (Segment 10 or 11) can be redirected to external data memory, as shown in **Figure 9-3**.

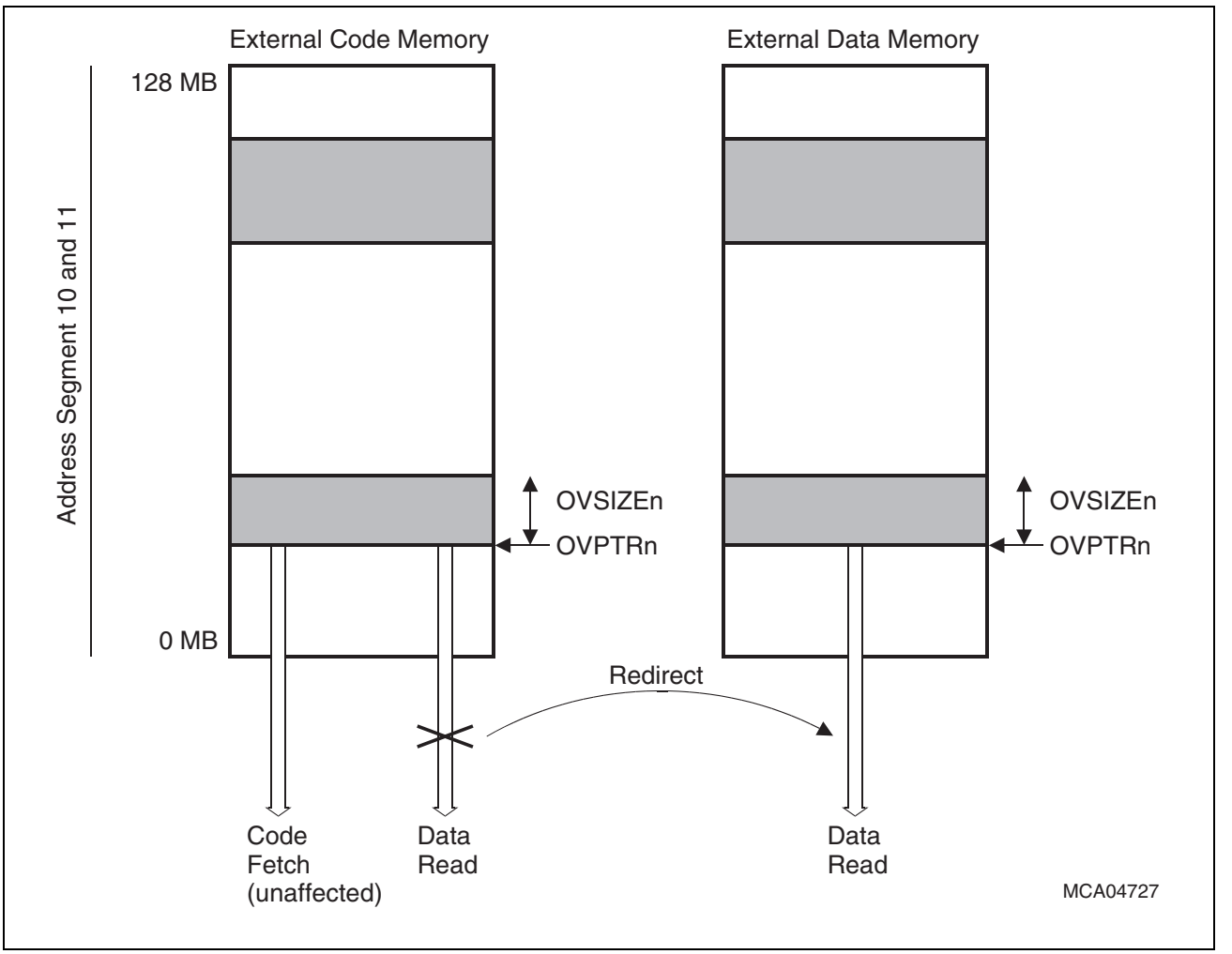


Figure 9-3 Redirection From External Code to External Data Memory

Two external overlay control registers (DMU_EOCRn, n = 0,1) control the redirection from external code memory to external data memory. Each register specifies the start address of an overlayable code memory area (DMU_EOCRn.OVPTR) and the size of the overlayable area (DMU_EOCRn.OVSIZE). The start address of each overlayable area must be located on addresses that are a multiple of the overlayable area's size. The external overlay memory is enabled by bit DMU_EOCRn.OEN. Bit field

DMU_EOCRn.SEG defines the segment of the external overlay area (valid segments: 10, 11 and 14).

The outgoing address on a data read from Segment 10 and 11 is checked against the addresses stored in the overlay address register. If the external overlay memory is enabled and an address match occurs, an access to the corresponding address in the external overlay memory is performed. Any of the external chip select signals can be programmed to be active on the overlayed access.

*Note: If external **and** internal overlay functionality is enabled and the address on a data fetch to the code memory matches an internal overlay area as well as an external overlay area, two redirected accesses are performed. The redirected access to the external overlay memory determines the access timing, while the data returned by the redirected access to the internal overlay memory is used.*

9.2.3 Redirection from External Code via $\overline{\text{CODE}}$ to External Data Memory

Data reads from external code memory can be redirected to external data memory, using the circuitry as shown in **Figure 9-4**.

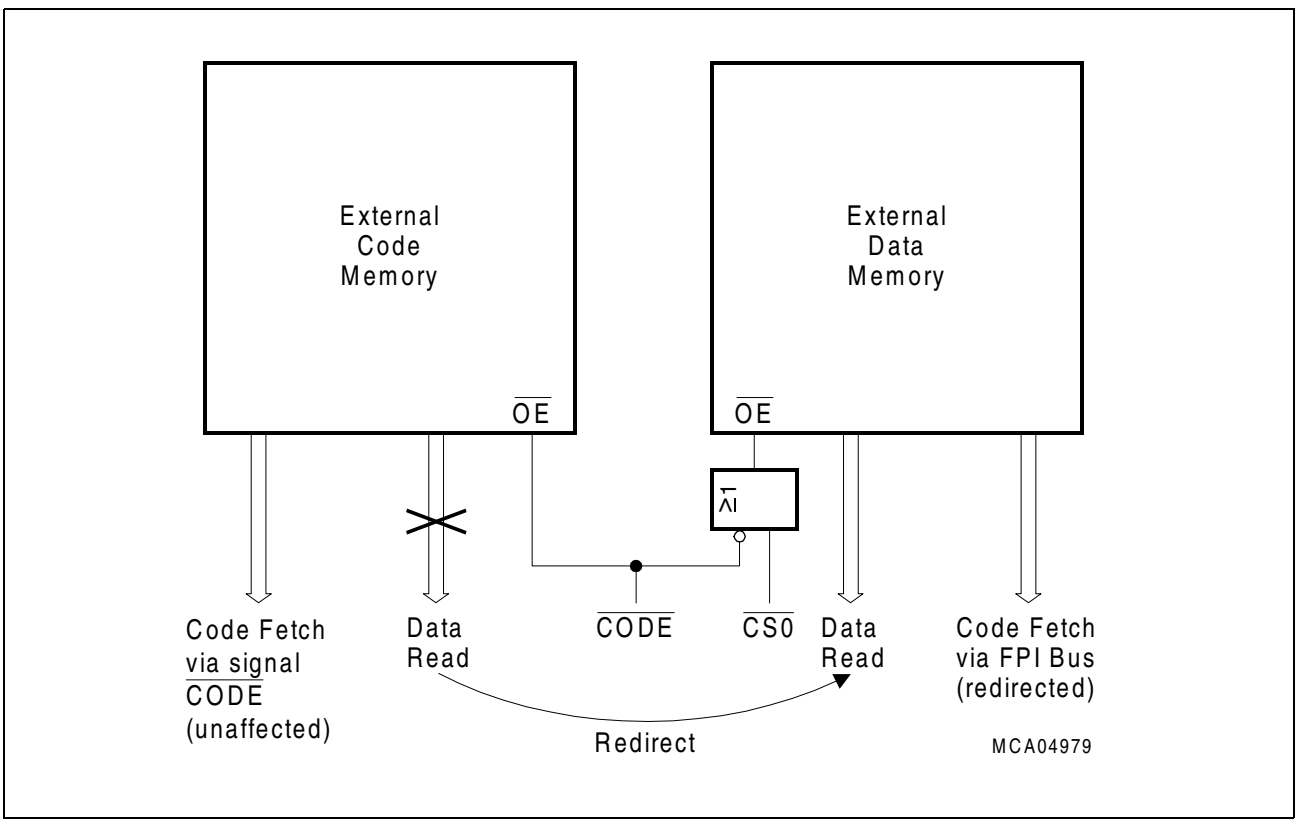


Figure 9-4 Redirection From External Code to External Data Memory

A data read from the code memory drives signal $\overline{\text{CS0}}$ active low ($\overline{\text{CS0}}$ = 0) and drives signal $\overline{\text{CODE}}$ inactive high ($\overline{\text{CODE}}$ = 1). The inverted $\overline{\text{CODE}}$ signal and the $\overline{\text{CS0}}$ signal

are logically or'ed and the data read from the code memory is redirected to the data memory. Note that for redirection from code to data memory, bit CS0D must be set in Register PMU_EIFCON.

The PMU burst mode instruction fetch is not affected by this redirection and will be performed from the code memory as normal. An instruction fetch via the FPI Bus is redirected to the data memory.

9.3 DMU Registers

As shown in Figure 9-5 and Table 9-3, one control register, two trap status registers, and several overlay register are implemented in the DMU. The registers and their bits are described in the following sections.

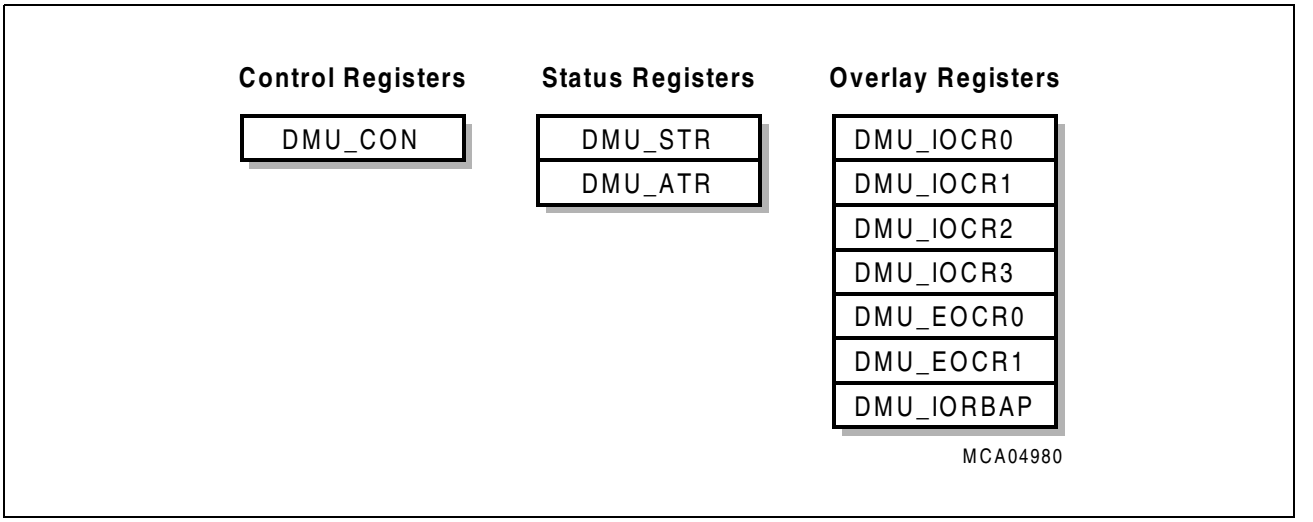


Figure 9-5 DMU Registers

Table 9-3 DMU Registers

Register Short Name	Register Long Name	Offset Address	Description see
DMU_CON	DMU Control Register	0010_H	Page 9-11
DMU_STR	DMU Synchronous Trap Flag Register	0018_H	Page 9-12
DMU_ATR	DMU Asynchronous Trap Flag Register	0020_H	Page 9-13
DMU_IOCR0	DMU Internal Overlay Control Register 0	0080_H	Page 9-14
DMU_IOCR1	DMU Internal Overlay Control Register 1	0088_H	
DMU_IOCR2	DMU Internal Overlay Control Register 2	0090_H	
DMU_IOCR3	DMU Internal Overlay Control Register 3	0098_H	
DMU_EOCR0	DMU External Overlay Control Register 0	$00A0_H$	Page 9-15
DMU_EOCR1	DMU External Overlay Control Register 1	$00A8_H$	
DMU_IORBAP	DMU Internal Overlay RAM Base Address Page Register	$00B8_H$	Page 9-17

Note: Accesses to DMU registers must be made with double-word-aligned word accesses. An access not conforming to this rule will cause a bus error if the access was from the FPI Bus, or a trap in case of a CPU load/store access.

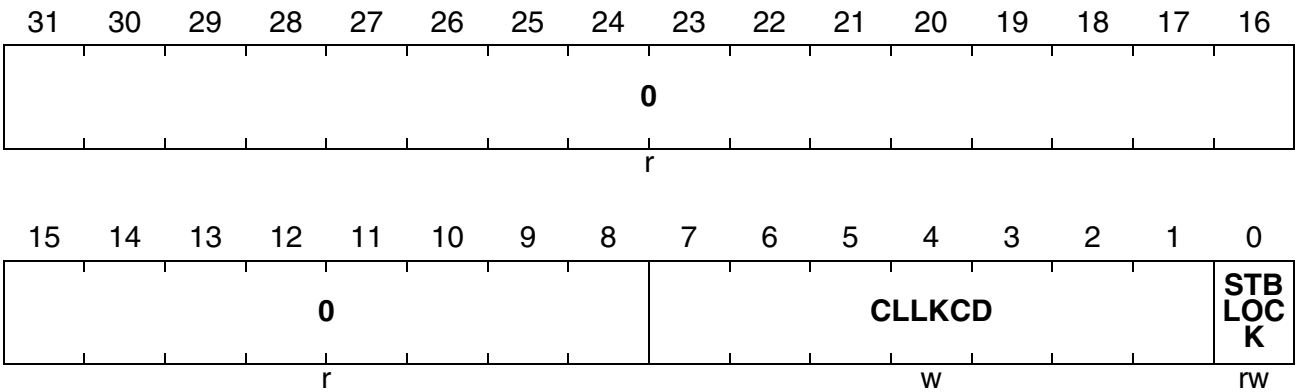
In the TC1765, the registers of the DMU are located in the following address range:

– Module Base Address. D7FF FF00$_H$
  Module End Address. D7FF FFFF$_H$
– Absolute Register Address = Module Base Address + Offset Address
  (offset addresses see **Table 9-3**)

9.3.1 Control Register

When the 32-KByte data memory of the TC1765 is used for standby operation it must be locked before it is put into the standby mode (standby power supplied via pin $V_{DDSBRAM}$). Otherwise data of the standby RAM can be corrupted during this operation.

DMU_CON
DMU Control Register **Reset Value: 0000 0000$_H$**

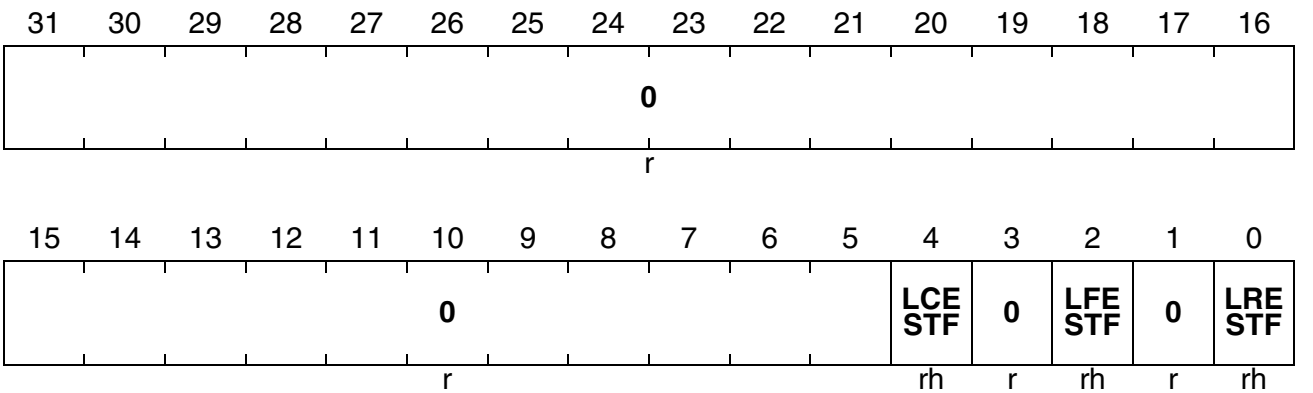


Field	Bits	Type	Description
STBLOCK	0	rw	**Lock Standby Data Memory** Bit can be set by writing a 1 to it (in supervisor mode only). Bit can only be reset (in supervisor mode only) by writing a 0 to it together with the appropriate clear lock code. 0 Normal operation of standby data memory 1 Standby data memory is locked. No read or write access of/to standby SRAM is possible. The state of STBLOCK after power-on reset is undefined. A hardware or software reset does not effect this bit.
CLLKCD	[7:1]	w	**Clear Lock Code** This bit field has to be written with 65$_H$ together with STBLOCK = 0 to reset the lock of the standby RAM.
0	[31:8]	r	**Reserved**; read as 0; should be written with 0.

9.3.2 Synchronous Trap Flag Register

The Synchronous Trap Flag Register, DMU_STR, holds the flags that inform about the root cause of a DMU Synchronous Trap (DSE) event.

DMU_STR
DMU Synchronous Trap Flag Register **Reset Value: 0000 0000$_H$**



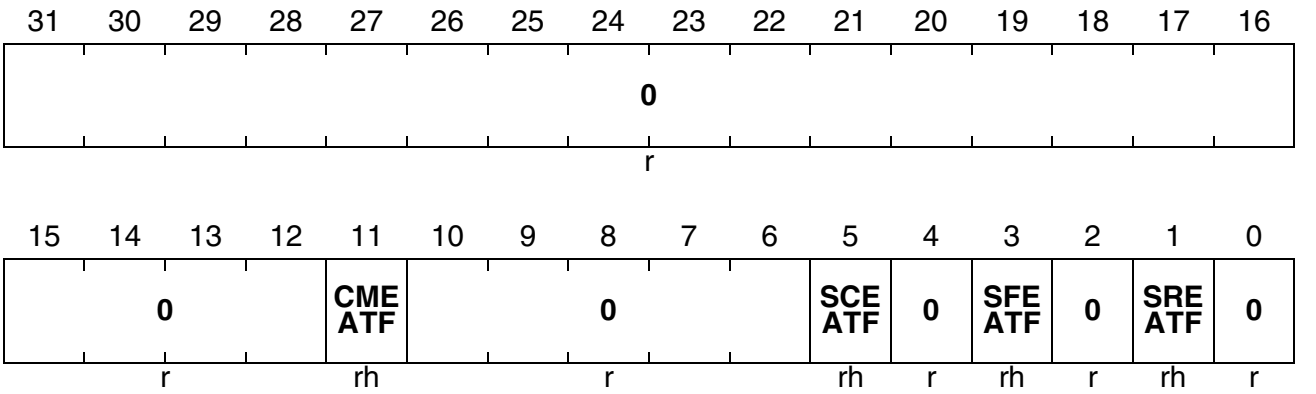
Field	Bits	Type	Description
LRESTF	0	rh	**Load Range Synchronous Error Flag** 0 No error 1 Synchronous load range error has occurred
LFESTF	2	rh	**FPI Bus Load Synchronous Error Flag** 0 No error 1 Synchronous FPI Bus load error has occurred
LCESTF	4	rh	**DMU Register Load Synchronous Error Flag** 0 No error 1 Synchronous DMU register load error has occurred
0	1, 3, [31:5]	r	**Reserved**; read as 0.

Note: When reading DMU_STR in Supervisor Mode, the contents of the register are returned and the bits of the register are then automatically cleared. Reading DMU_STR in User Mode returns the contents only, the register is not cleared.

9.3.3 Asynchronous Trap Flag Register

The Asynchronous Trap Flag Register, DMU_ATR, holds the flags that inform about the root cause of a DMU Asynchronous Trap (DAE) event.

DMU_ATR
DMU Asynchronous Trap Flag Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SREATF	1	rh	**Store Range Asynchronous Error Flag** 0 No error 1 Asynchronous error has occurred
SFEATF	3	rh	**FPI Bus Store Asynchronous Error Flag** 0 No error 1 Asynchronous error has occurred
SCEATF	5	rh	**DMU Register Store Asynchronous Error Flag** 0 No error 1 Asynchronous error has occurred
CMEATF	11	rh	**Cache Management Asynchronous Error Flag** 0 No error 1 Asynchronous error has occurred *Note: see also* ***Section 9.1.4***
0	0, 2, 4, [10:6], [31:12]	r	**Reserved**; read as 0.

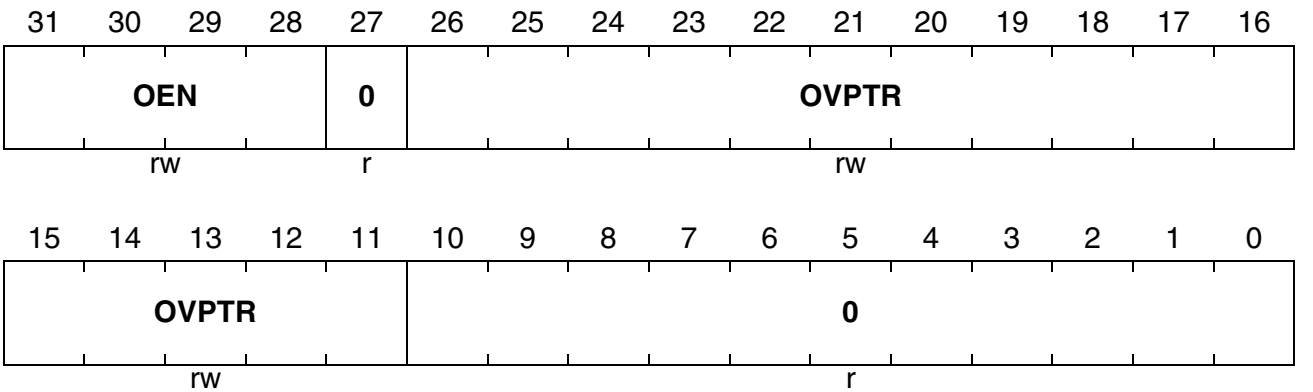
Note: When reading DMU_ATR in Supervisor Mode, the contents of the register are returned and the bits of the register are automatically cleared. Reading DMU_ATR in User Mode returns the contents only, the register is not cleared.

9.3.4 Overlay Functionality Registers

In the TC1765, there are four internal overlay control registers. Each DMU_IOCR register specifies the start address of an overlayed address space within Segment 10 and Segment 11. If bit OEN is 0, the appropriate overlay area is disabled. The address range of the external code memory that can be overlaid is 128 MBytes. A data read access to an overlaid external code memory area is redirected to the specified address within the internal data memory (Segment 13).

The start address within the overlay control register layout is specified in such a way that the real address inside Segment 13 may be generated without any shifting.

DMU_IOCRn (n = 3-0)
DMU Internal Overlay Control Register n **Reset Value: 0000 0000$_H$**



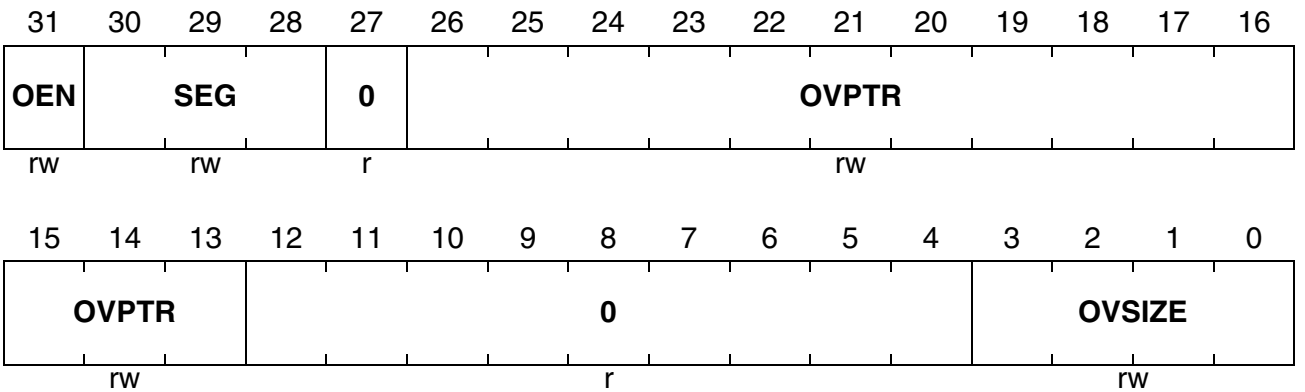
Field	Bits	Type	Description
OVPTR	[26:11]	rw	**Code Memory Overlay Area n Start Address Pointer** This address pointer specifies the 2K start address of the overlayable area n located in the external code memory.
OEN	[31:28]	rw	**Enable Code Memory Overlay RAM Area n** This bit field enables the internal overlay function for the external code memory overlayable area n. 0000_B Overlay function is disabled 1010_B Overlay function is enabled others Reserved; do not use these combinations;
0	[10:0], 27	r	**Reserved**; read as 0; should be written with 0.

Note: The DMU_IOCRn registers can be only read/written using 32-bit accesses.

There are two external overlay control registers. Each DMU_EOCR register specifies the start address and size of an overlaid memory block within Segment 10 and Segment 11. A data read access to such an overlaid memory block can be redirected to external data memory located either in Segment 10, 11 or 14.

A data fetch to an externally overlaid region results in a FPI Bus access with an altered segment number and bit 27 forced to 0.

DMU_EOCRn (n = 0, 1)
DMU External Overlay Control Register n **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
OVSIZE	[3:0]	rw	**Overlayable Memory Size** This value defines the size of the overlayed external code memory block n. **OVSIZE** **Block Size** 0000_B 8 KBytes 0001_B 16 KBytes 0010_B 32 KBytes 0011_B 64 KBytes 0100_B 128 KBytes 0101_B 256 KBytes 0110_B 512 KBytes 0111_B 1 MByte 1000_B 2 MBytes 1001_B 4 MBytes 1010_B 8 MBytes 1011_B 16 MBytes 1100_B 32 MBytes 1101_B 64 MBytes 1110_B 128 MBytes 1111_B 128 MBytes

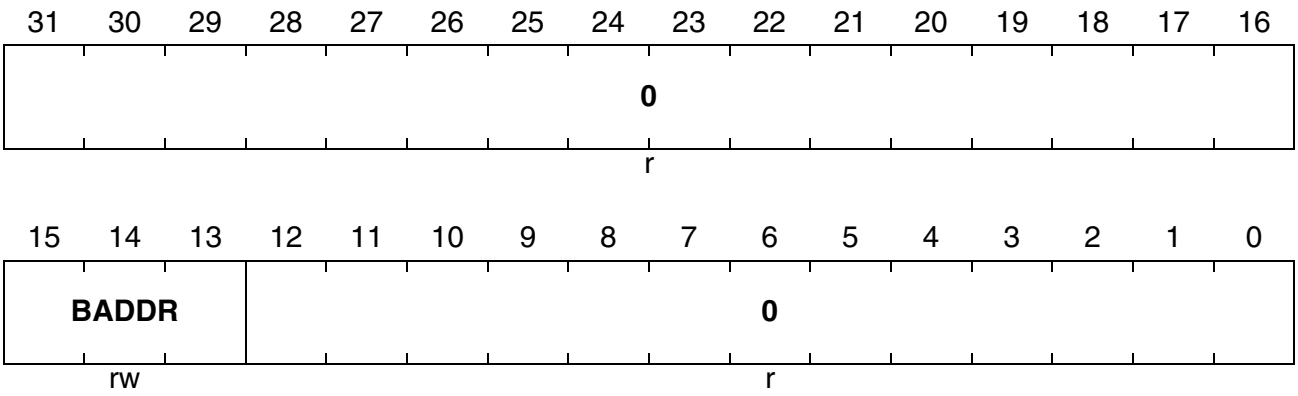
Field	Bits	Type	Description
OVPTR	[26:13]	rw	**Overlay Area n Start Address Pointer** This address pointer specifies the start address of the overlayable area n within the external code memory which is redirected. The start address of an overlayable area must be a multiple of OVSIZE.
SEG	[30:28]	rw	**External Memory Segment Selection** Defines the external memory segment which is used for external overlay. 010_B Redirection to Segment 10 011_B Redirection to Segment 11 110_B Redirection to Segment 14 others Reserved; leads to unpredictable results.
OEN	31	rw	**Enable External Overlay for Code Memory Area n** Enables the external overlay function for the Code Memory overlayable area n. 0 Overlay function is disabled 1 Overlay function is enabled
0	[12:4], 27	r	**Reserved**; read as 0; should be written with 0.

Note: The DMU_EOCRn registers allow 32-bit access only.

The DMU_IORBAP register is used to define the start address of the 8-KByte overlay RAM block, located in the internal data RAM.

DMU_IORBAP
DMU Internal Overlay RAM Base Address Page Register

Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
BADDR	[15:13]	rw	**Internal Overlay RAM Page Base Address** This address pointer specifies the base address of the 8-KByte overlay RAM block located in the internal data RAM.
0	[12:0], [31:20]	r	**Reserved**; read as 0; should be written with 0.

Note: The DMU_IORBAP register allows 32-bit access only.

10 Direct Memory Access Controller (DMA)

This chapter describes the Direct Memory Access (DMA) Controller of the TC1765. This chapter contains the following sections:

- Functional description of the DMA Kernel (see **Section 10.1**).
- Register descriptions for all DMA Kernel specific registers (see **Section 10.2**).
- TC1765 implementation specific details and registers of the DMA Controller, including control, interrupt control, address decoding, and clock control (see **Section 10.3**).

*Note: The DMA Kernel register names described in **Section 10.2** will be referenced in the TC1765 User's Manual with the module name prefix "DMA_" for the DMA interface.*

10.1 DMA Controller Kernel Description

The Direct Memory Access (DMA) Controller executes DMA transactions from a source address location to a destination address location, without intervention of the CPU. One DMA transaction is controlled by one DMA channel. Each DMA channel has assigned its own two channel register sets: a shadow register set and an active register set. This feature eases programming and releases the CPU from control tasks.

Features:

- 8 independent DMA channels (4 per DMA block)
  - 4 DMA selectable request inputs per DMA channel
  - Fixed priority of DMA channels within a DMA block
  - Software and hardware DMA request generation
- Support of FPI Bus to FPI Bus DMA transactions
- Individually programmable operation modes for each DMA channel
  - Single mode: stops and disables DMA channel after a predefined number of DMA transfers
  - Continuous mode: DMA channel remains enabled after a predefined number of DMA transfers; DMA transaction can be repeated;
- Full 32-bit addressing capability of each DMA channel
  - 4 GByte address range
  - Source and destination transfer individually programmable in steps from 0 to 255 bytes
  - Support of circular buffer addressing mode
- Programmable data width of a DMA transaction: 8-bit, 16-bit, or 32-bit
- Register set for each DMA channel
  - Source and destination start address register
  - Source and destination end address register
  - Channel control and status register
  - Offset and transfer count register
- Bus bandwidth allocation
- Flexible interrupt generation

10.1.1 Definition of Terms

DMA Transfer

A *DMA transfer* is an operation which consists always of two parts:

– A *source transfer* which loads data from a data source into the DMA controller
– A *destination transfer* which puts data from the DMA controller to a data destination

Within a *DMA transfer* data is always transferred from the data source via the DMA controller to the data destination. The data width of *source transfer* and *destination transfer* are always identical (8-bit, 16-bit, or 32-bit).

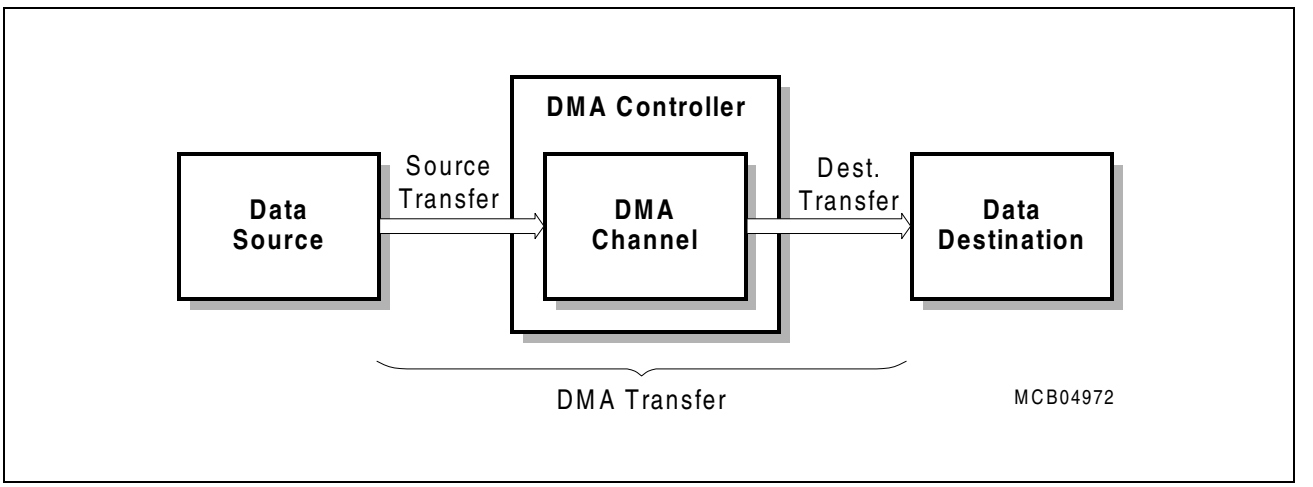


Figure 10-1 DMA Terms Definitions

DMA Transaction

A *DMA transaction* is composed of several (at least by one) *DMA transfers*. The *Transfer Count* defines the number of *DMA transfers* within one *DMA transaction*.

10.1.2 DMA Principle

The DMA controller supports DMA transfers from FPI Bus to FPI Bus. DMA Transfers can be requested either by hardware or by software. DMA hardware requests are triggered by specific request lines from the peripheral modules (see **Figure 10-2**). The number of available DMA request lines from a peripheral module varies depending on the module functionality. Typically, the occurrence of a receive or transmit data interrupts in a peripheral module can generate in parallel to the interrupt request a DMA request. Therefore, the interrupt control unit and the DMA controller in the TC1765 can react independently on interrupt and DMA requests that have been generated by one source.

The DMA controller consists of a control unit and two DMA blocks. The control unit includes a FPI Bus slave interface for programming of the DMA controller registers. Once configured, each block of the DMA controller is able to act as a master on the FPI Bus using its FPI Bus master interface.

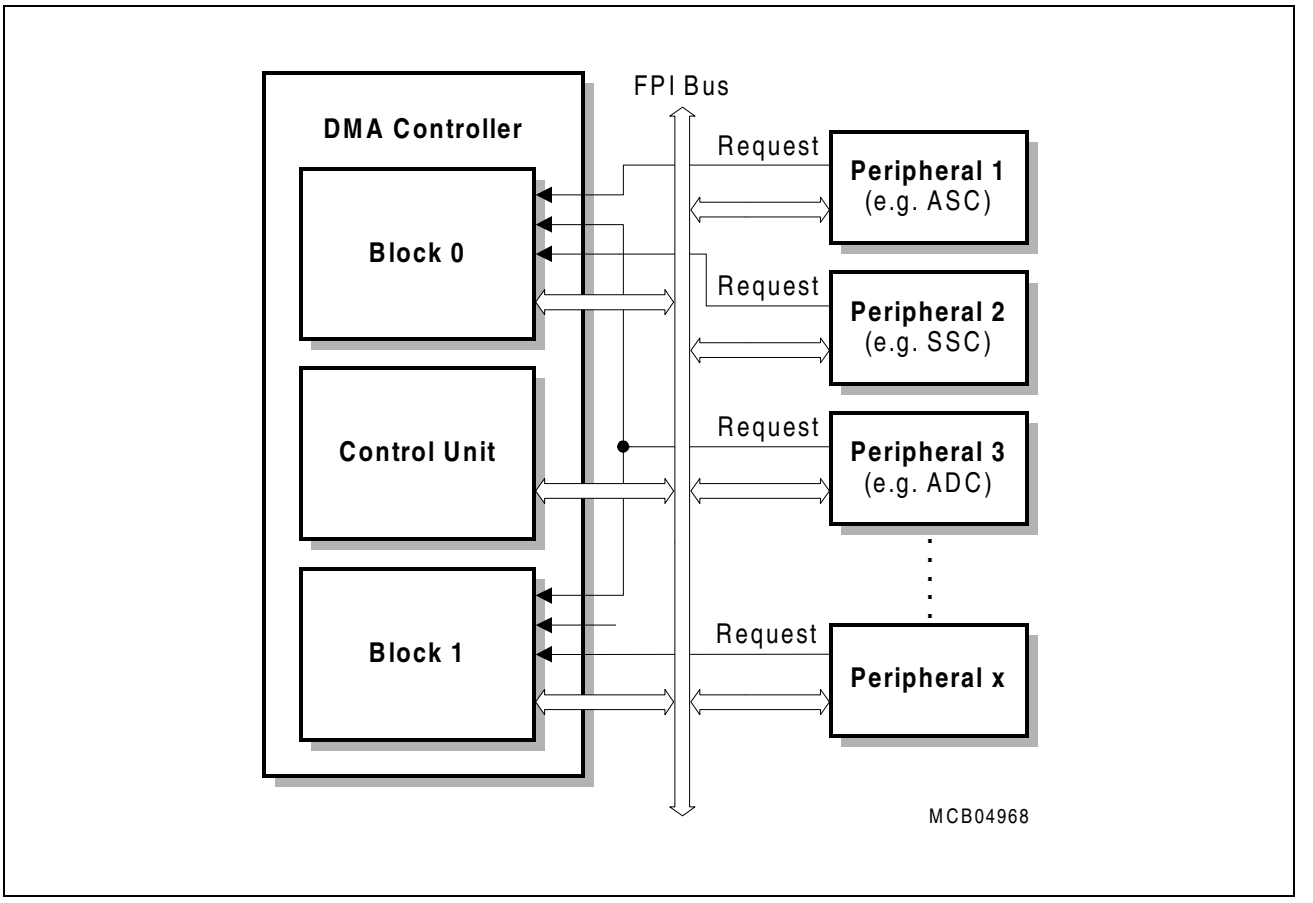


Figure 10-2 DMA Principle

To ease the architecture of the DMA controller, one data buffer per each DMA channel is implemented. Neither block nor split modes are supported on the FPI Bus.

10.1.3 DMA Block Diagram

Each of the two blocks in the DMA controller, block 0 and block 1, provides four DMA channels with sixteen DMA request inputs. The request assignment unit in each block assign one DMA request input to each DMA channel. The control unit includes a third request unit dedicated especially for request control through I/O pins. This unit is connects two of eight request inputs with two request outputs which can be then wired externally of the DMA controller module to the request inputs of the two DMA controller blocks. Request assignment unit 2 evaluates pulses or levels by its edge detect and level select logic.

Clock control, address decoding, and interrupt service request control for the eight interrupt service request outputs are managed outside the DMA controller module kernel.

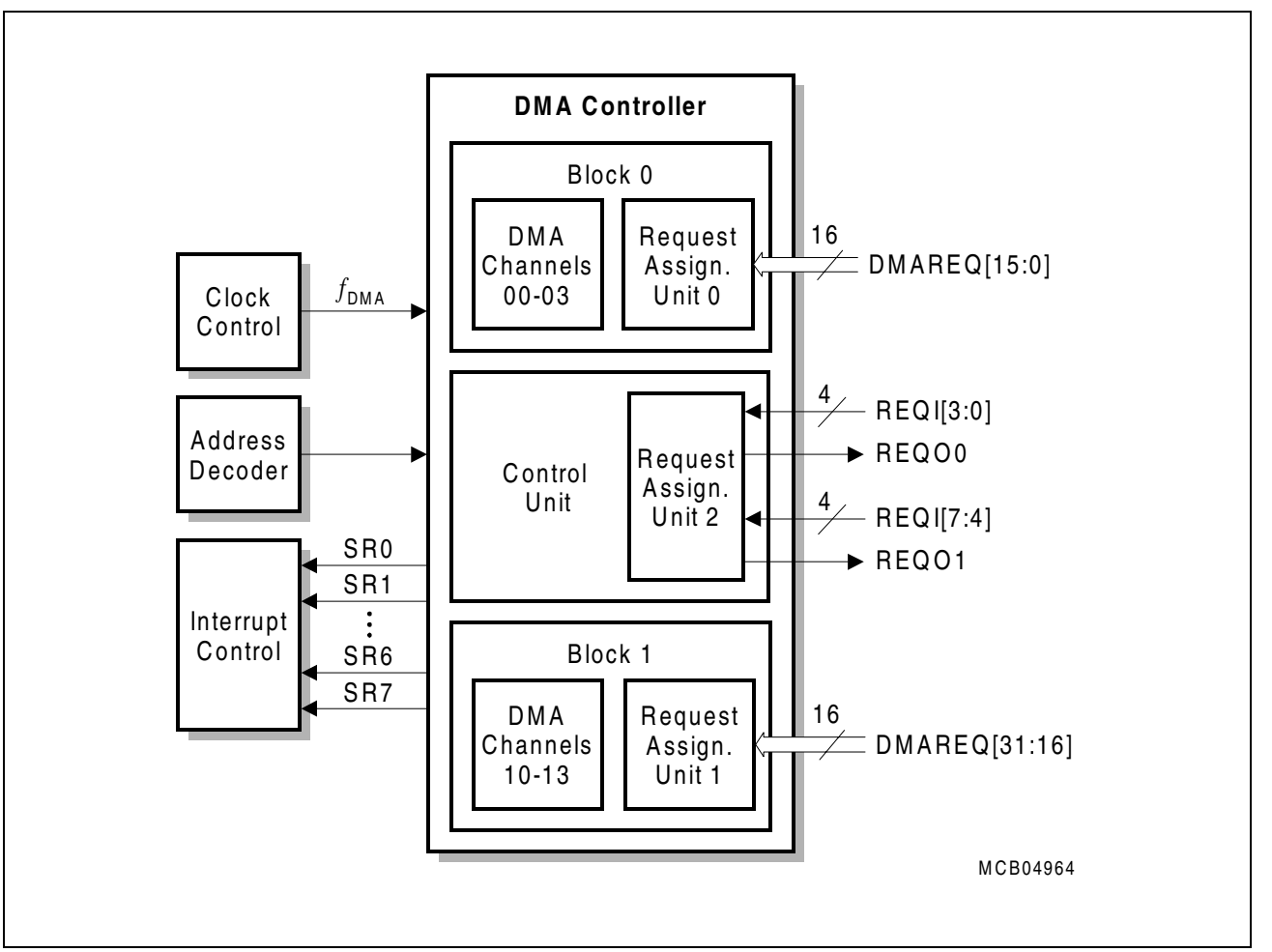


Figure 10-3 Block Diagram of DMA Controller Module

Each DMA transaction consists of a programmable number of DMA transfers that can be either 8-bit, 16-bit or 32-bit wide. Data is read from the source (source transfer) and stored intermediately in the corresponding DMA channel data buffer. In the second part of the DMA transfer, data is written from the DMA channel data buffer to the destination

(destination transfer). No data assembling or disassembling functionality is provided by the DMA controller.

10.1.4 DMA Operation Functionality

Each DMA channel has two register sets: one active register set and one shadow register set (see **Figure 10-4**). These register sets are also named as “header”. Only the shadow register set can be read or written by the CPU. After programming of the shadow register set, a DMA transaction is started after the shadow register set has been transferred to the active register set. Each register of a DMA channel except the actual transfer count status and status flags in control and status register CSRn is duplicated (shadowed). The actual transfer count information during an active DMA transaction can be read back in register CSRn as status information.

A DMA transaction is initiated either by software (immediately started after the header transfer operation) or by hardware via the DMA request input CHn_REQ. After completion of a DMA transaction, a service request signal is generated to the service request node of DMA channel n.

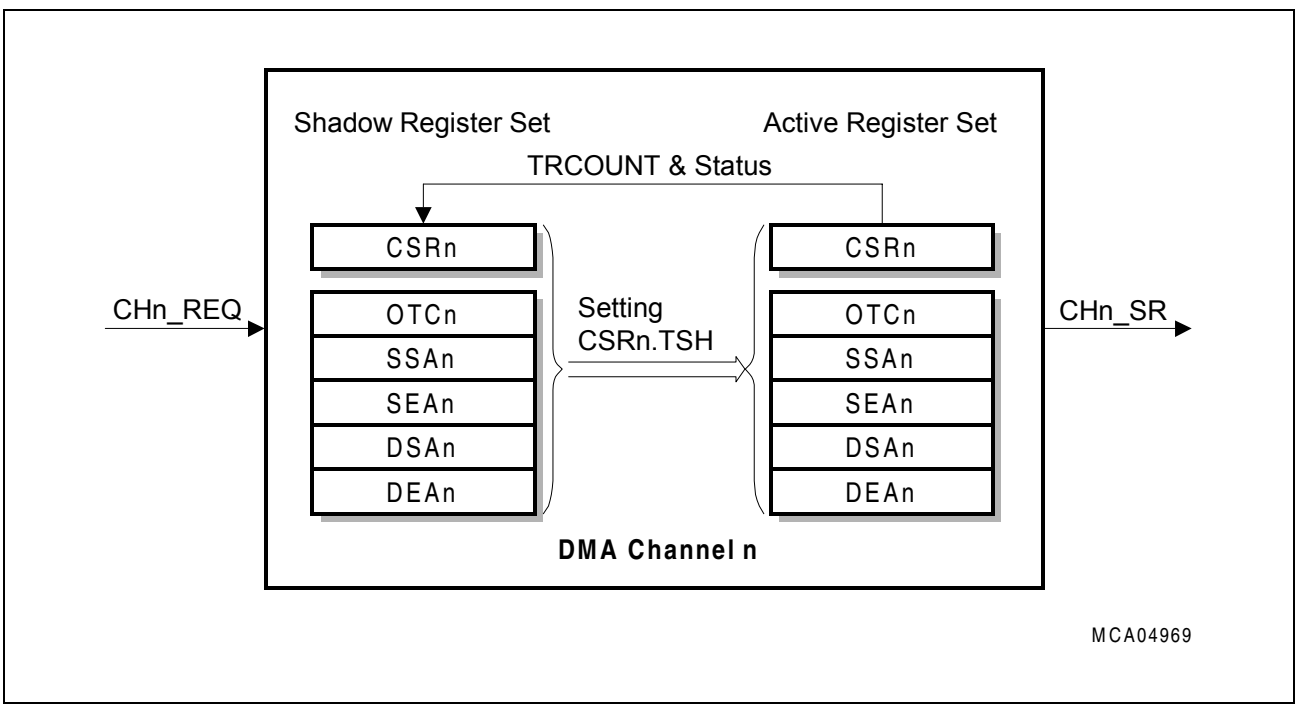


Figure 10-4 Register Interface of a DMA Channel

10.1.5 DMA Channel Configuration

For the setup of a DMA channel, the following steps must be performed:

1. The registers of the shadow register set OTCn, SSAn, SEAn, DSAn, DEAn of the DMA channel must be configured.
2. The transfer shadow header bit CSRn.TSH is set to transfer the shadow register set with all DMA transaction parameters to the active register set and to enable the DMA channel for DMA transaction. Of course, when writing to the CSRn with TSH set, also the channel transfer control bit CHTC, the channel operation mode bit CHMODE, the channel data width bit CHDW, and the peripheral request select bit field PRSEL (in hardware controlled modes only) are transferred to the active register set.
   If the actual DMA channel is currently active, the shadow header transfer operation is delayed until the end of this DMA transaction (bit CSRn.TSH is still set). CSRn.TSH is automatically cleared by hardware when the shadow register set has been transferred to the active register set.

10.1.5.1 DMA Transfer Triggering

There are two ways of triggering a DMA transfer within a DMA transaction of a DMA channel. The two trigger modes are controlled by the channel transfer control bit CSRn.CHTC.

1. CSRn.CHTC = 0: a DMA transfer of DMA channel n starts automatically as soon as the active register set has been loaded by the content of the shadow register set.
2. CSRn.CHTC = 1: a DMA transfer of DMA channel n waits for an active DMA request input as selected by CSRn.PRSEL.

10.1.5.2 DMA Channel Operation Mode

The operation mode is individually programmable for each DMA channel n via control bit CSRn.CHMODE. A channel can operate either in one of two modes:

- Single mode or
- Continuous mode

In single mode, the DMA channel n active status bit CSRn.CHAC is cleared after the last DMA transfer of a DMA transaction in order to disable this DMA channel n. For the start of the next DMA transaction, DMA channel n transfer shadow header bit CSRn.TSH must be set again.

In continuous mode, the DMA channel n active status bit CSRn.CHAC remains set, which allows DMA requesting units to initiate subsequent DMA transactions without any CPU intervention.

Software Controlled Single Mode

This mode in selected by CSRn.CHTC = 0 and CSRn.CHMODE = 0.

In software controlled single mode, setting of CSRn.TSH causes the shadow header to be transferred to the active header and the DMA transaction to be started. The DMA transaction consists of a predefined number of DMA transfers (tc), as defined in OTCn.TRCOUNT. When OTCn.TRCOUNT reaches 0000_H, the DMA channel n becomes disabled and the interrupt service line SRn is activated. Setting CSRn.TSH again starts the next DMA transaction with the parameters as defined in the shadow register set. The running DMA transaction is indicated by channel active flag CSRn.CHAC set.

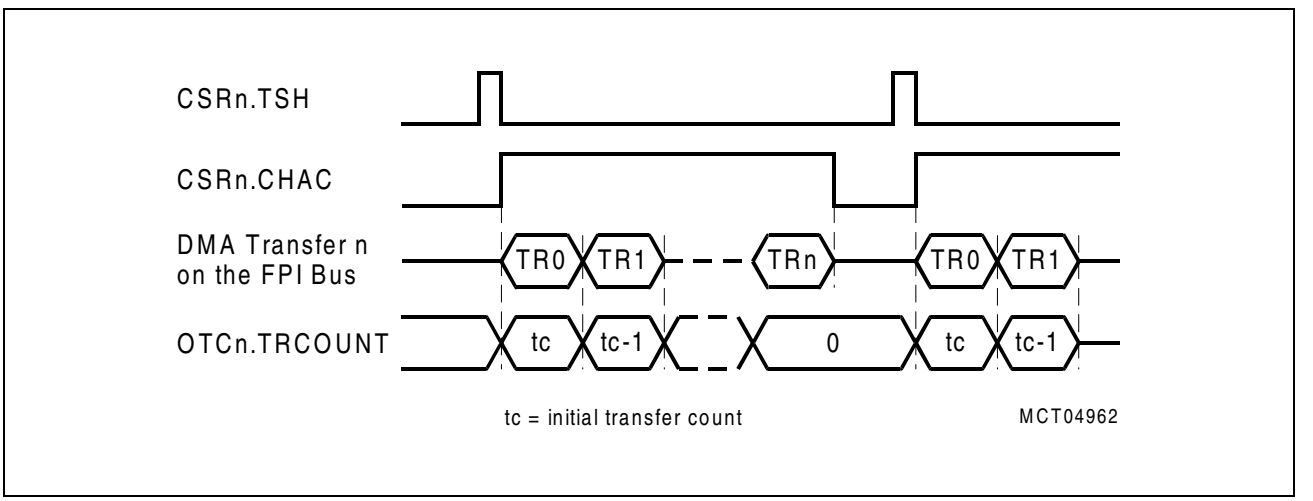


Figure 10-5 Software Controlled Single Mode Operation

Software Controlled Continuous Mode

This mode is selected by CSRn.CHTC = 0 and CSRn.CHMODE = 1.

In software controlled continuous mode, setting of CSRn.TSH causes the shadow header to be transferred to the active header and the DMA transaction to be started. The DMA transaction consists of a predefined number of DMA transfers (tc), as defined in OTCn.TRCOUNT. Each time when OTCn.TRCOUNT reaches 0000_H, the interrupt service line SRn of DMA channel n is activated and DMA channel starts a new DMA transaction with the parameters as defined in the shadow register set. When bit CSRn.CHMODE is cleared, the current DMA transaction of DMA channel n is finished after proceeding until OTCn.TRCOUNT = 0000_H, and the interrupt service line SRn is activated. During continuous mode the active flag CSRn.CHAC is always set.

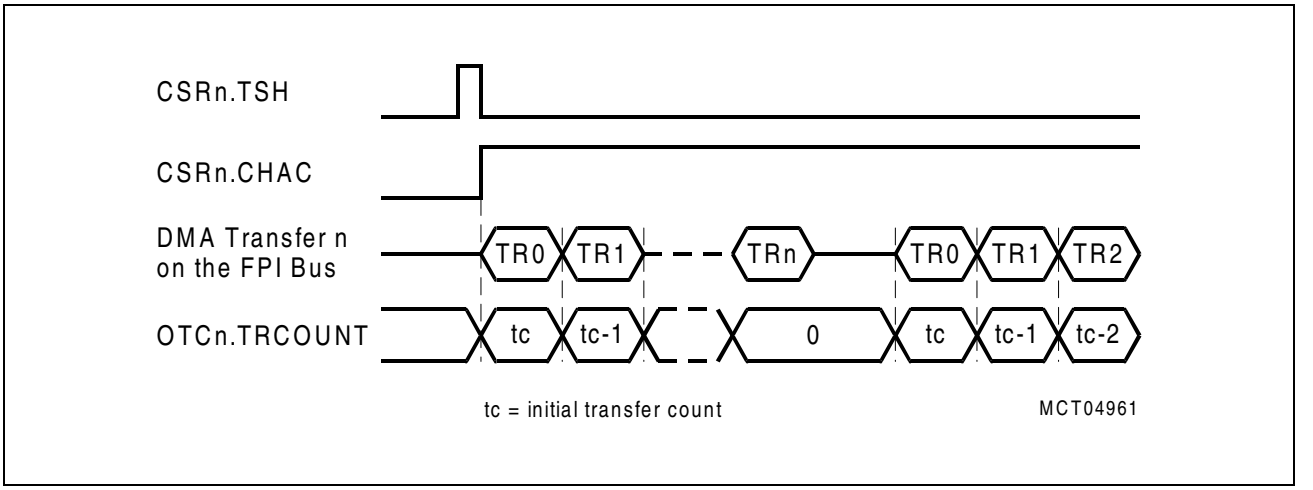


Figure 10-6 Software Controlled Continuous Mode Operation

Hardware Controlled Single Mode

This mode is selected by CSRn.CHTC = 1 and CSRn.CHMODE = 0.

In hardware controlled single mode, setting CSRn.TSH causes the shadow header to be transferred to the active header and the DMA transaction to be started. The DMA transaction consists of a predefined number of DMA transfers (tc), as defined in OTCn.TRCOUNT. A DMA transfer of the DMA transaction is executed whenever the DMA request input line of DMA channel n becomes active. When OTCn.TRCOUNT reaches 0000_H, the DMA channel n becomes disabled (CSRn.CHAC = 0) and the interrupt service line SRn is activated. Setting CSRn.TSH again starts the next DMA transaction with the parameters as defined in the shadow register set. The running DMA transaction is indicated by channel active flag CSRn.CHAC set.

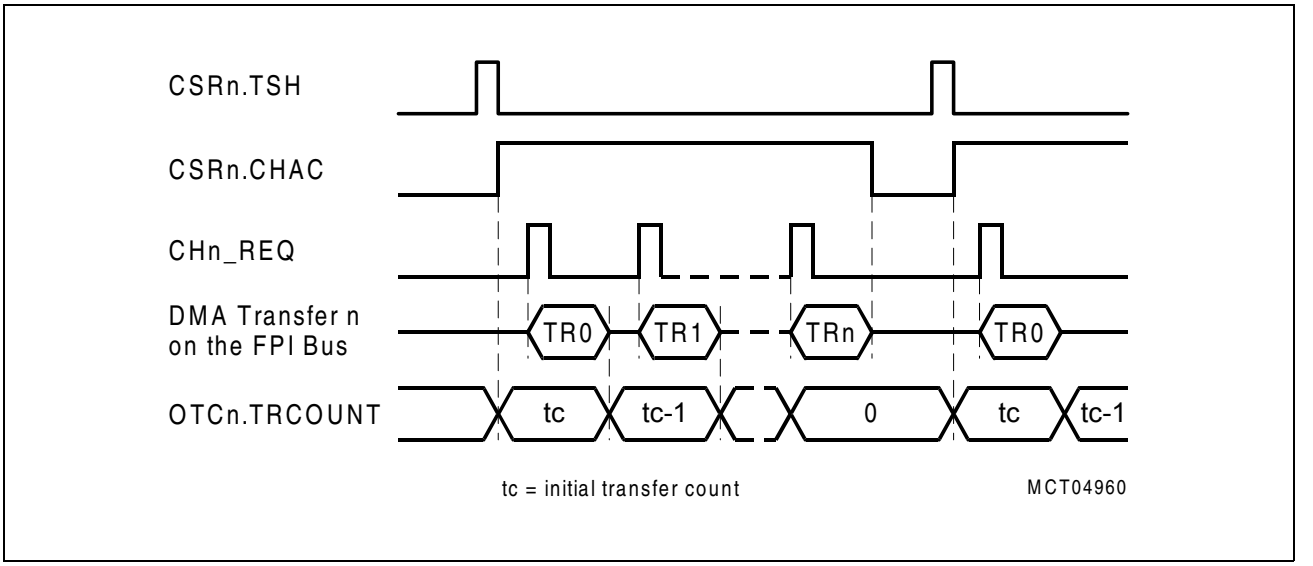


Figure 10-7 Hardware Controlled Single Mode Operation

Hardware Controlled Continuous Mode

This mode is selected by CSRn.CHTC = 1 and CSRn.CHMODE = 1.

In hardware controlled continuous mode, setting of CSRn.TSH causes the shadow header to be transferred to the active header and the DMA transaction to be started. The DMA transaction consists of a predefined number of DMA transfers (tc), as defined in OTCn.TRCOUNT. A DMA transfer of the DMA transaction is executed whenever the DMA request input line of DMA channel n becomes active. Each time when OTCn.TRCOUNT reaches 0000_H, the interrupt service line SRn of DMA channel n is activated and DMA channel starts a new DMA transaction with the parameters as defined in the shadow register set. When bit CSRn.CHMODE is cleared, the current DMA transaction of DMA channel n is finished after proceeding until OTCn.TRCOUNT = 0000_H, and the interrupt service line SRn is activated. During continuous mode the active flag CSRn.CHAC is always set.

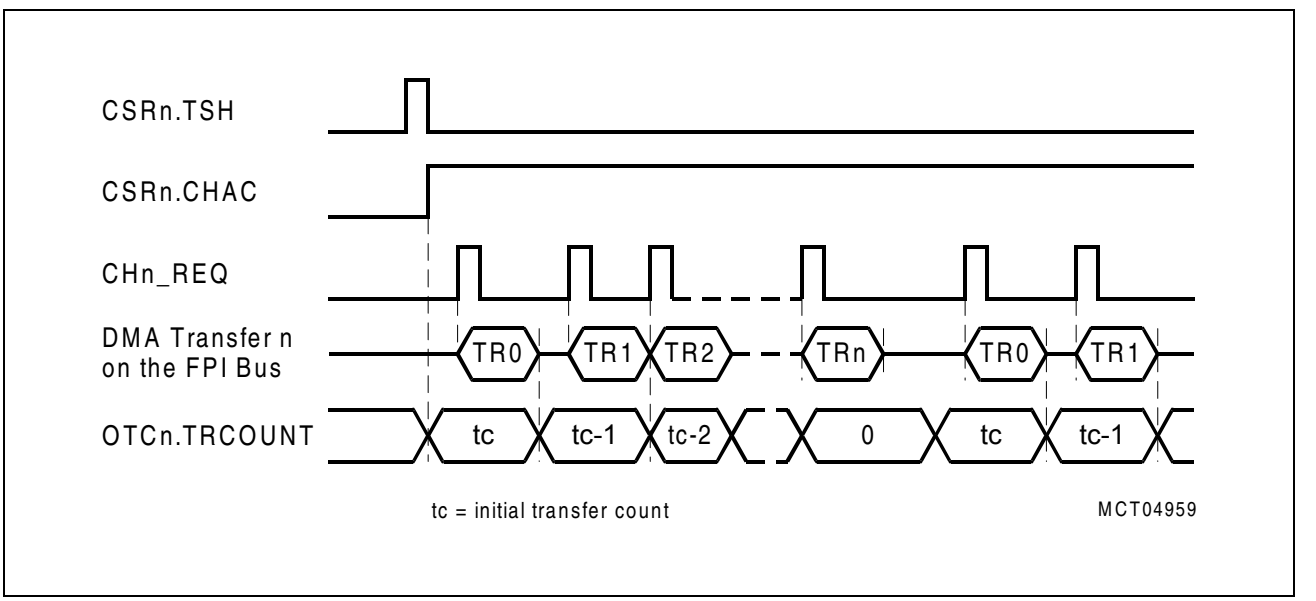


Figure 10-8 Hardware Controlled Continuous Mode Operation

10.1.5.3 DMA Transfer Parameters

Transfer Count

One parameter of a DMA transaction is the 16-bit transfer count. This 16-bit transfer count defines the number of DMA transfers to be executed within one DMA transaction. Its value is defined by the number of requested DMA transfers decremented by 1. This means for example, that a 0000_H must be programmed for the transfer count when one DMA transfer should be executed.

Address Calculation Principles

Besides the transfer count, each part of a DMA transfer, source and destination transfer, is controlled by three parameters:

- 32-bit Start Address (SA), stored in registers SSAn and DSAn
- 32-bit End Address (EA), stored in registers SEAn and DEAn
- 8-bit Offset (OFF), stored in bit fields SRCOFS and DESTOFS of register OTCn

The address for the next source/destination transfer is calculated according Section 10-9.

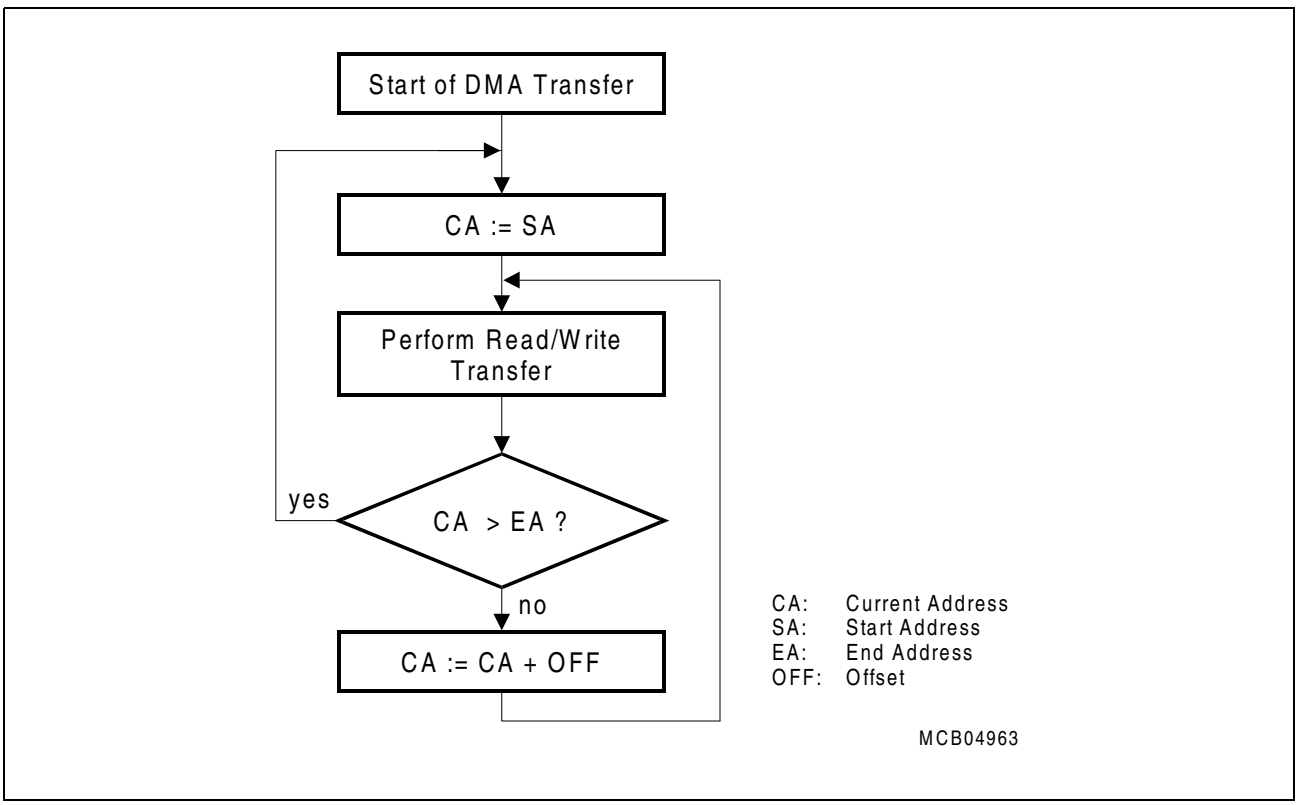


Figure 10-9 Source/Destination Transfer Address Update

In continuous mode, the end address for source or destination transfers is calculated by

- End address = Address of the last transfer - offset field in OTCn

Example for a DMA Transaction Parameter Calculation

For a DMA transaction with 4 word transfers, source and destination increment, the following parameters must be programmed:

OTCn.SRCOFS	= 04_H	(= 4 byte source address increment)
OTCn.DESTOFS	= 04_H	(= 4 byte destination address increment)
OTCn.TRCOUNT	= 0003_H	(= requested number of transfers - 1)
CSRn.CHDW	= 10_B	(= 32-bit transfers)

10.1.5.4 DMA Bus Bandwidth Limitation

After each source transfer or destination transfer the related DMA block will release the FPI Bus for at least 3 FPI Bus clock cycles. This gives other FPI Bus masters the chance to arbitrate for the FPI Bus ownership.

10.1.5.5 Error Conditions

The DMA controller allows to detect error conditions individually for each DMA channel.

The source transfer error flag CSRn.SRCERR flag indicates an FPI Bus error that occurred during a source transfer (read) of a DMA transaction. The destination transfer error flag CSRn.DESTERR flag indicates an FPI Bus error that occurred during a destination transfer (write) of a DMA transaction.

The request overrun error flag CSRn.REQOVR indicates if DMA requests for a DMA channel have been lost. Each DMA channel has a DMA request counter, which counts up to 15 DMA requests in hardware controlled modes. The DMA request counter is incremented with each incoming DMA request and decremented after each destination (write) transfer. With the overflow condition of the DMA request counter (counter value = 15 and DMA request occurs) bit CSRn.REQOVR is set while the counter value remains set to 15.

10.1.5.6 Channel Reset Operation

A DMA transfer of a DMA channel n can be stopped by setting bit SCRn.CHRST. When an FPI Bus access of DMA channel n is just running during setting of SCRn.CHRST, this FPI Bus access is terminated before the DMA channel can be restarted again. This behavior guarantees the occurrence of data consistency.

A user program should execute the following steps for resetting and restarting a DMA channel:

1. Writing a 1 to SCRn.CHRST.
2. Polling bit SCRn.CHAC until bit is set to 0.
3. Restarting the DMA channel n again by writing a 1 to SCRn.TSH.

10.1.6 Request Assignment Units 0 and 1

Each DMA block, block 0 and 1, contains one request assignment unit, that multiplexes the sixteen request inputs of a DMA block to one request input for each DMA channel. A pulse on the DMAREQn lines takes at least two clock cycles. Each request input multiplexer of DMA channel n is controlled by the peripheral request select bit field CSRn.PRSEL. The DMA channels of a DMA block have assigned a fixed priority: the DMA channel with the lowest number has the highest priority. The priority of the DMA blocks among each other is defined by the assignment of its FPI Bus master interfaces as FPI Bus master agent.

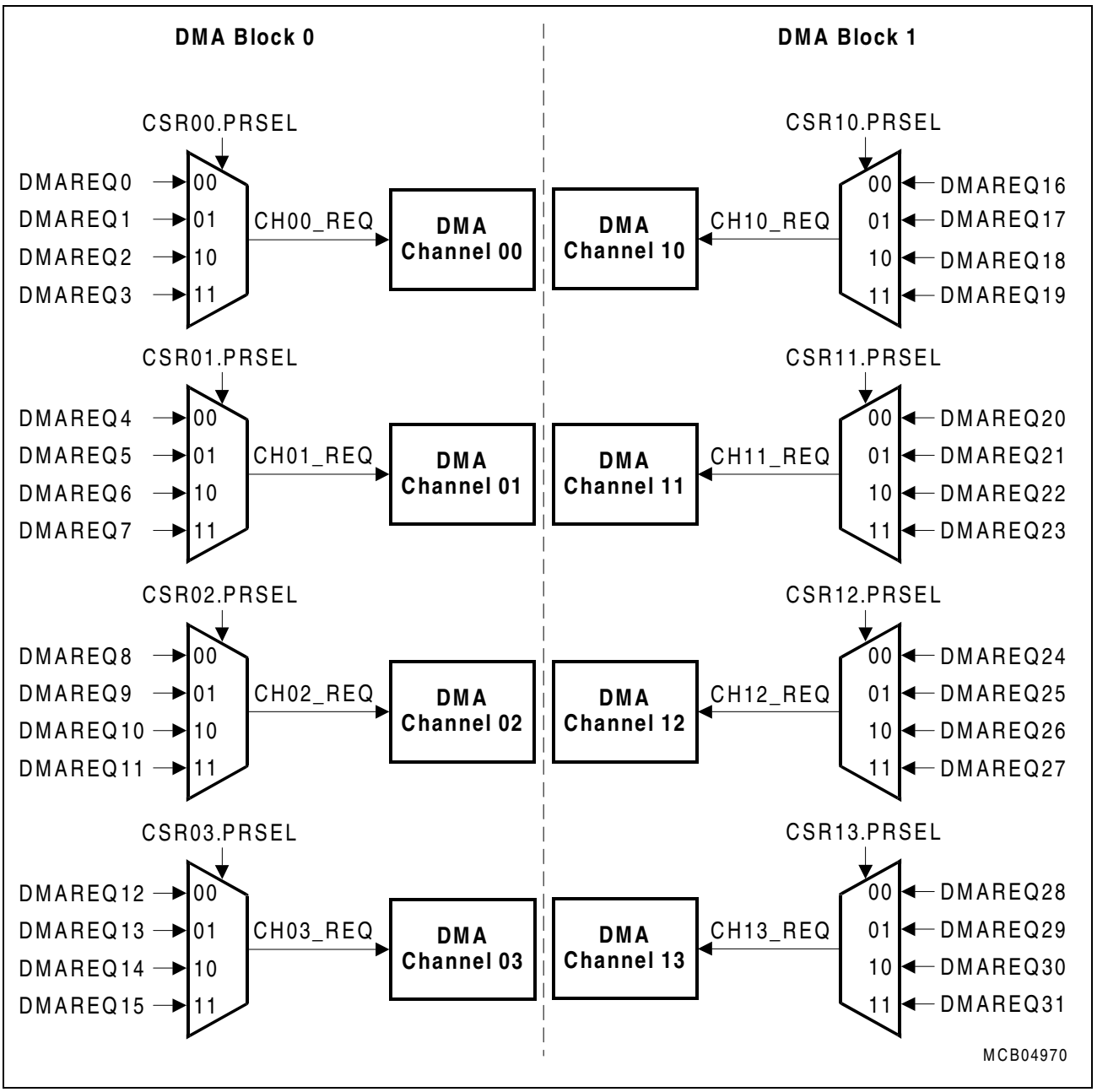


Figure 10-10 Request Assignment Unit in DMA Block 0 and 1

10.1.7 Request Assignment Unit 2

Besides the two DMA request assignment units in each DMA block, the DMA controller has a request assignment unit 2 which is located in the control unit of the DMA controller. Request assignment unit 2 allows to control external DMA transfer requests coming from an I/O pin by providing edge detection and level select logic (gating functionality). The two outputs REQO0 and REQO1 are typically wired externally of the DMA controller kernel back to DMA request input lines of DMA block 0 or 1. The functionality of the request assignment unit 2 is controlled by the global control register GCTR.

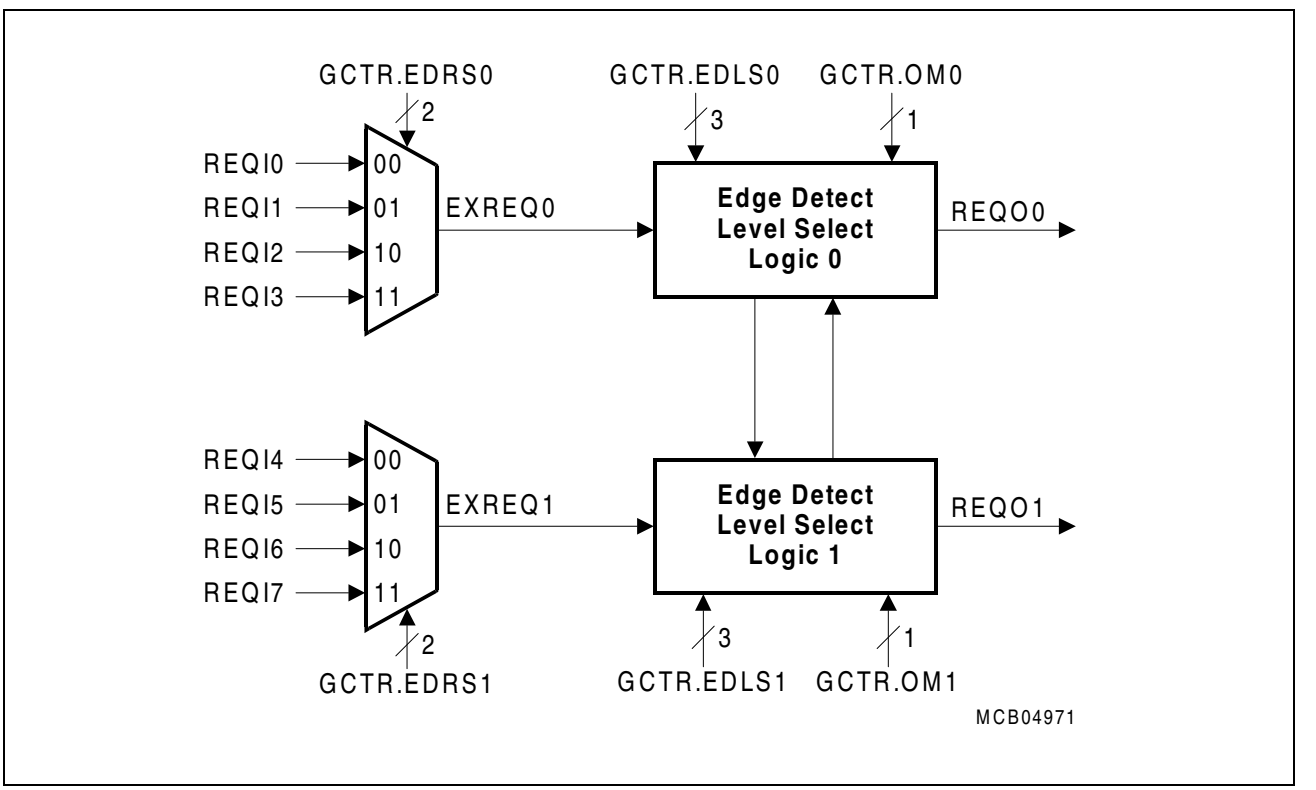


Figure 10-11 Request Assignment Unit 2 Block Diagram

An input multiplexer selects one of four possible request input lines. Each edge detect level select logic has one request output assigned. The two edge detection and level select logics in the request assignment unit 2 provide in general two modes, selected by bit GTCR.OMx:

- Edge sensitive request mode
- Level sensitive (gating) mode

In edge sensitive mode, the edge detect level select logic 0 generates a pulse of two clock cycles width at its output signal REQO0 whenever a rising and/or falling signal transition occurs at the input signal EXREQ0. Edge detect level select logic 1 must be programmed for edge sensitive mode, too. Programmed for level sensitive mode, input EXREQ1 of edge detect level select logic 1 operates as a gating input for the REQO0 output. The gating level can be programmed to be active on low or high level. The

equivalent operation is possible for edge detect level select logic 0 when edge detect level select logic 1 is operating in edge sensitive mode.

As a summary, the two edge detect level select logic can operate both in edge sensitive mode (independently from each other) or both in opposite modes (dependently from each other). **Table 10-1** shows the possible combinations of operation modes.

Table 10-1 Request Assignment Unit 2 Modes

Mode of		Input Operation		Output Operation	
Edge Detect Level Select Logic 0	**Edge Detect Level Select Logic 1**	**EXREQ0**	**EXREQ1**	**REQO0**	**REQO1**
Edge sensitive OM0 = 0	Edge sensitive OM1 = 0	Request input	Request input	Request output	Request output
	Level sensitive OM1 = 1		Gating input for EXREQ0		–
Level sensitive OM0 = 1	Edge sensitive OM1 = 0	Gating input for EXREQ1	Request input	–	Request output
	Level sensitive OM1 = 1[1)]	–	–	–	–

1) This combination of GCTR.OM0 and GCTR.OM1 should not be used, because in this case, both outputs, REQO0 and REQO1, are inoperable.

10.2 DMA Module Kernel Registers

Figure 10-12 and Table 10-2 show all registers associated with the DMA Kernel.

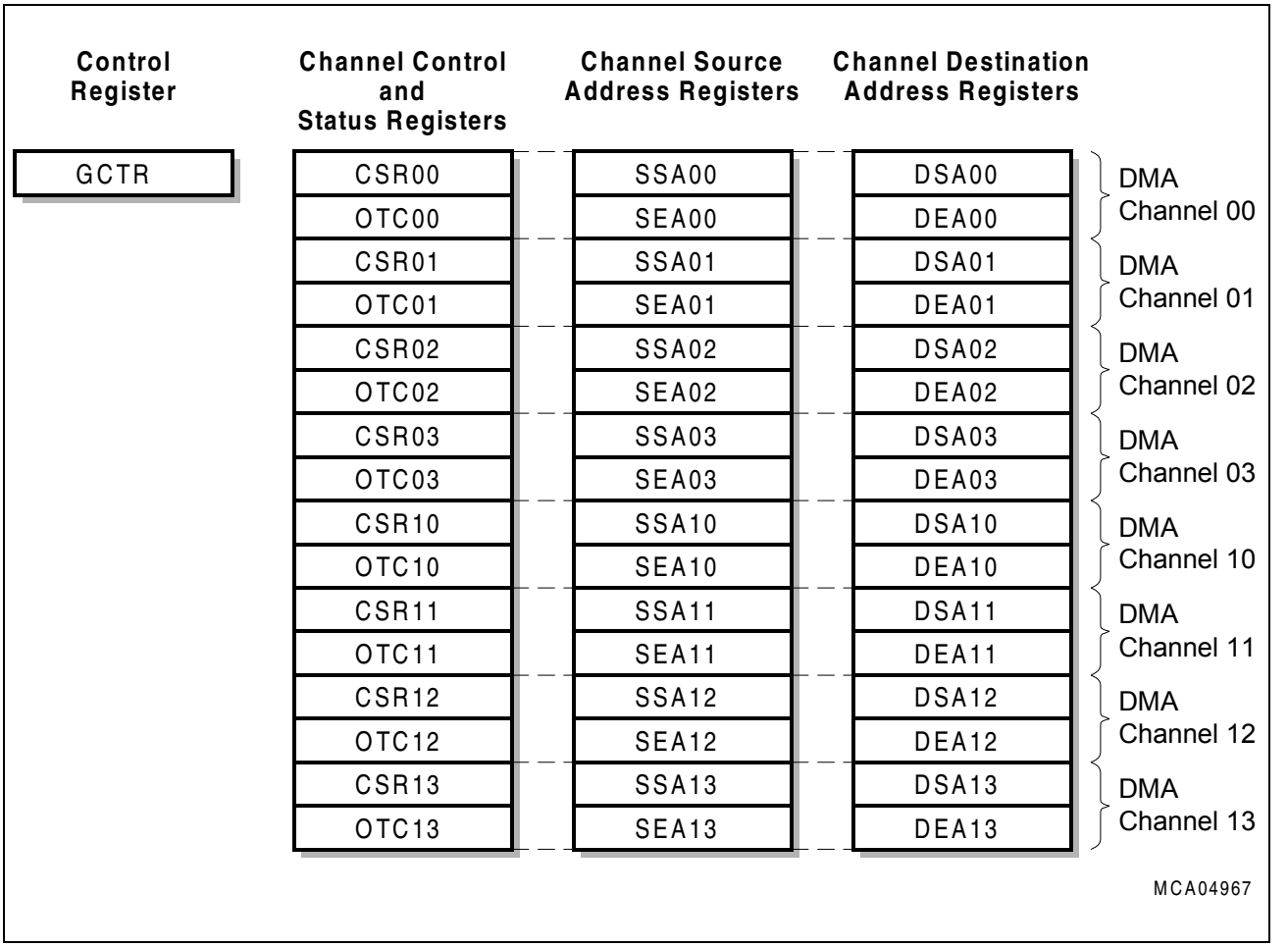


Figure 10-12 DMA Kernel Registers

Table 10-2 DMA Kernel Registers

Register Short Name	Register Long Name	Offset Address	Description see
GCTR	DMA Global Control Register	0010_H	Page 10-19
CSRn	DMA Channel n Control and Status Register	$m \times 20_H + 00_H$	Page 10-22
OTCn	DMA Channel n Offset and Transfer Count Register	$m \times 20_H + 04_H$	Page 10-25
SSAn	DMA Channel n Source Start Address Register	$m \times 20_H + 08_H$	Page 10-26
SEAn	DMA Channel n Source End Address Register	$m \times 20_H + 0C_H$	Page 10-26
DSAn	DMA Channel n Destination Start Address Register	$m \times 20_H + 10_H$	Page 10-27
DEAn	DMA Channel n Destination End Address Register	$m \times 20_H + 14_H$	Page 10-27

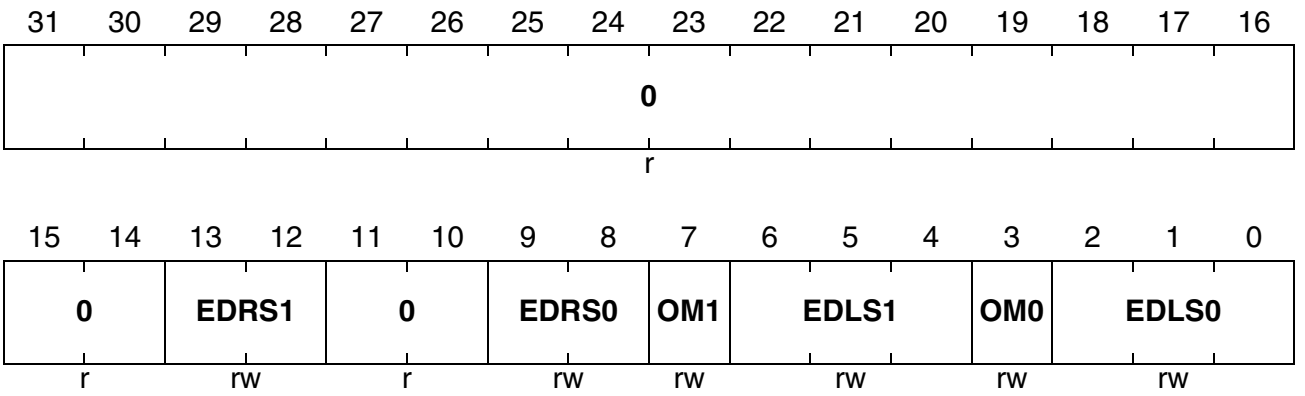
Table 10-3 Offset Address Factor Assignment DMA Channel n to m

n	m	n	m
00	1	10	5
01	2	11	6
02	3	12	7
03	4	13	8

Note: All DMA kernel register names described in this section will be referenced in other parts of the TC1765 User's Manual with the module name prefix "DMA_" for the DMA interface.

The global control register contains the control bits for the request control unit 2 of the DMA module. See also **Section 10.1.7** for more details of the external request control unit.

GCTR
Global Control Register **Reset Value: 0000 0000$_H$**



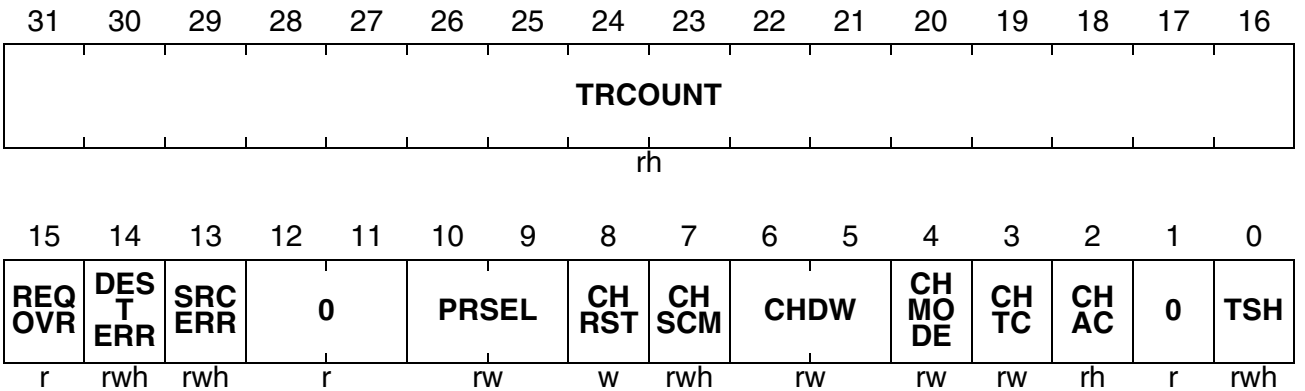
Field	Bits	Type	Description
EDLS0	[2:0]	rw	**Edge Detect Level Select Logic 0 Control** This bit field together with bit OM0 controls the operating mode of the edge detect level select logic 0. **OM0 = 0 (OM1 = 0): Edge sensitive request mode** 000_B No action 001_B Detect falling edges at EXREQ0 to REQO0 010_B Detect rising edges at EXREQ0 to REQO0 011_B Detect rising and falling edges at EXREQ0 to REQO0 $1XX_B$ Reserved **OM0 = 1 (OM1 = 0): Level sensitive (gating) mode** $0XX_B$ Reserved 100_B EXREQ0 = 0 enables REQO1 for DMA request pulse generation (gating of REQO1) 101_B EXREQ0 = 1 enables REQO1 for DMA request pulse generation (gating of REQO1) $11X_B$ Reserved

Field	Bits	Type	Description
OM0	3	rw	**Operation Mode of Edge Detection and Level Select Logic 0** This bit defines the operating mode of the edge detect level select logic 0. 0 Edge sensitive request mode selected 1 Level sensitive (gating) mode selected; output REQO0 cannot be used for DMA requests.
EDLS1	[6:4]	rw	**Edge Detect Level Select Logic 1 Control** This bit field together with bit OM1 controls the operating mode of the edge detect level select logic 1. **OM1 = 0 (OM0 = 0): Edge sensitive request mode:** 000_B No action 001_B Detect falling edges at EXREQ1 to REQO1 010_B Detect rising edges at EXREQ1 to REQO1 011_B Detect rising and falling edges at EXREQ1 to REQO1 $1XX_B$ Reserved **OM1 = 1 (OM0 = 0): Level sensitive (gating) mode** $0XX_B$ Reserved 100_B EXREQ1 = 0 enables REQO0 for DMA request pulse generation (gating of REQO0) 101_B EXREQ1 = 1 enables REQO0 for DMA request pulse generation (gating of REQO0) $11X_B$ Reserved
OM1	7	rw	**Operation Mode of Edge Detection and Level Select Logic 1** This bit defines the operating mode of the edge detect level select logic 1. 0 Edge sensitive request mode selected 1 Level sensitive (gating) mode selected; output REQO1 cannot be used for DMA requests.
EDRS0	[9:8]	rw	**External DMA Request Select Control 0** This bit field controls the input multiplexer 0 of the external request control unit. 00_B REQI0 is selected 01_B REQI1 is selected 10_B REQI2 is selected 11_B REQI3 is selected

Field	Bits	Type	Description
EDRS1	[13:12]	rw	**External DMA Request Select Control 1** This bit field controls the input multiplexer 1 of the external request control unit. 00_B REQI4 is selected 01_B REQI5 is selected 10_B REQI6 is selected 11_B REQI7 is selected
0	[31:14] [11:10]	r	**Reserved**; returns 0 if read; should be written with 0.

The control and status register assigned to each DMA channel contains its control and status flags.

CSRn (n = 00-03 and n = 10-13)
Control and Status Register **Reset Value: 0000 0000$_H$**



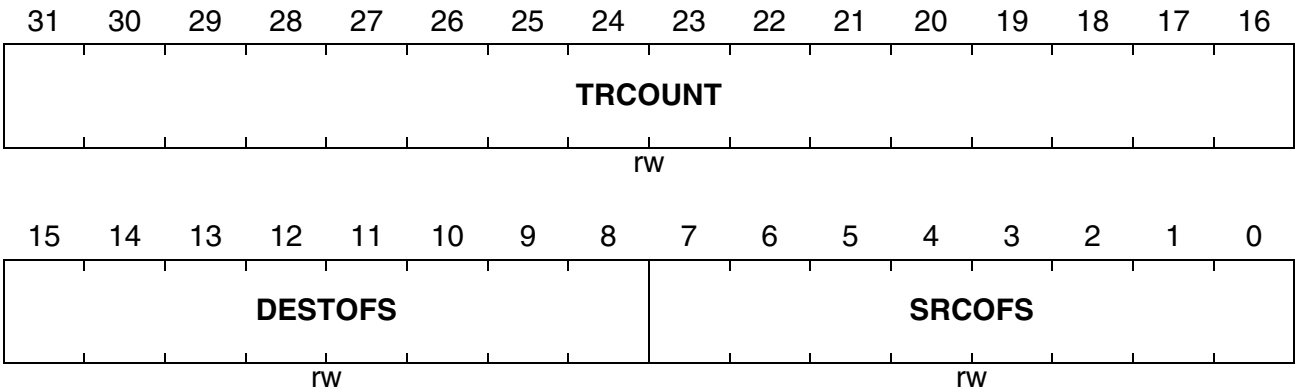
Field	Bits	Type	Description
TSH	0	rwh	**Transfer Shadow Header and Enable Channel n** This bit controls the transfer of the shadow header for DMA channel n and starts the DMA channel n for DMA transfer. 0 No action 1 Transfer the shadow header of DMA channel n to the active header of DMA channel n, as soon as the corresponding channel is inactive, and enable the DMA channel n for DMA transfer afterwards. TSH remains set until the shadow header has been transferred to the active header. After this transfer TSH is reset by hardware. TSH can be reset by software, too.
CHAC	2	rh	**Channel Active Flag** This bit is set by hardware at least two DMA clock cycles after TSH has been set by a CSRn write operation. CHAC remains set as long as DMA channel n is performing a DMA transfer. 0 DMA channel n is inactive 1 DMA channel n is active Any write to this bit has no effect.

Field	Bits	Type	Description
CHTC	3	rw	**Channel Transfer Control** This bit specifies whether the transfer rate in a DMA Transfer is controlled by hardware (a DMA requesting source) or by software. 0 The transfer rate in a DMA transfer is controlled by software, assuming that the source and destination locations are ready. 1 The transfer rate in a DMA transfer is controlled by the corresponding channel request line of the DMA requesting source.
CHMODE	4	rw	**Channel Operation Mode** This bit field defines the operating mode of DMA channel n. 0 Single mode operation selected for DMA channel n 1 Continuous mode operation selected for DMA channel n
CHDW	[6:5]	rw	**Channel Data Width** CHDW specifies the data width for source and destination transactions of DMA channel n. 00_B 8-bit (byte) transfers selected 01_B 16-bit (half-word) transfers selected 10_B 32-Bit (word) transfers selected 11_B Reserved; don't use this combination.
CHSCM	7	rwh	**Channel Stop Continuous Mode** Setting CHSCM forces the DMA channel n continuous mode to be changed from continuous to single mode at the end of the DMA transaction. The channel acts afterwards as if single mode was initially selected and the channel operation mode bit CSRn.CHMODE was cleared. 0 No action 1 Stop continuous mode at the end of the DMA transaction.

Field	Bits	Type	Description
CHRST	8	w	**Channel Reset** This bit forces DMA channel n to stop its current DMA transfer and resets all bits in CSRn (except bit field TRCOUNT). 0 No action 1 Stop DMA channel n and reset CSRn bits. Bit is always read as 0.
PRSEL	[10:9]	rw	**Peripheral Request Select** This bit field controls the input multiplexer of DMA channel n in the request assignment unit. 00 Multiplexer input 0 selected 01 Multiplexer input 1 selected 10 Multiplexer input 2 selected 11 Multiplexer input 3 selected
SRCERR	13	rwh	**Source Transfer Error Flag** This bit is set whenever an FPI Bus error occurred during a source (read) transfer of a DMA transaction executed at DMA channel n. Bit is reset by writing a 0 to this bit location.
DESTERR	14	rwh	**Destination Write Operation Error Flag** This bit is set whenever an FPI Bus error occurred during the destination (write) transfer of a DMA transaction executed at DMA channel n. Bit is reset by writing a 0 to this bit location.
REQOVR	15	rwh	**Request Overrun Error Flag** This bit is set whenever an overrun of DMA requests occurs on DMA channel n. Bit is reset by writing a 0 to this bit location.
TRCOUNT	[31:16]	rh	**Transfer Count Status** This bit field contains the actual value of the DMA transfer count of an active DMA transaction at DMA channel n.
0	1, [12:11]	r	**Reserved**; returns 0 if read; should be written with 0.

The offset and transfer count register contains the source and destination offset as well as the transfer count value.

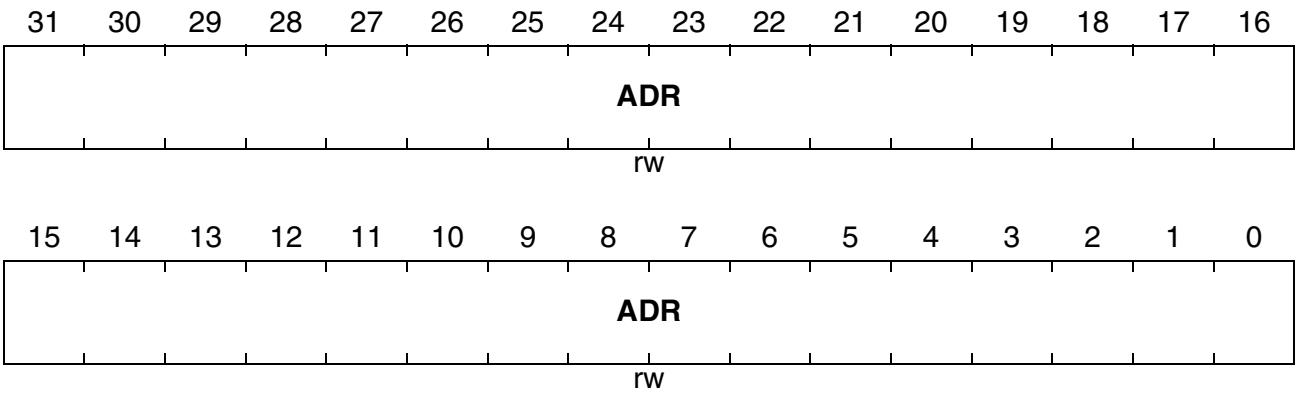
OTCn (n = 00-03 and n = 10-13)
Offset and Transfer Count Register n **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SRCOFS	[7:0]	rw	**Source Offset** This bit field specifies the address offset in bytes to be added to the source address pointer, after a read transfer from the source buffer.
DESTOFS	[15:8]	rw	**Destination Offset** This bit field specifies the address offset in bytes to be added to the destination address pointer, after a write transfer to the destination buffer.
TRCOUNT	[31:16]	rw	**Transaction Counter** This bit field contains the number of DMA transfers to be performed within one DMA transaction decremented by 1.

The source start address register contains the 32-bit start address of the source buffer.

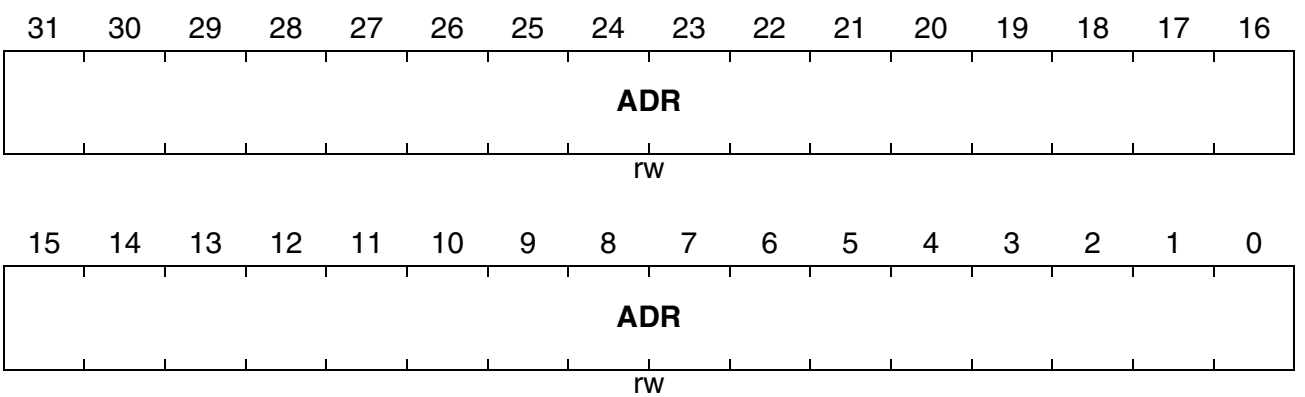
SSAn (n = 00-03 and n = 10-13)
Source Start Address Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ADR	[31:0]	rw	**Source Start Address** This bit field specifies the 32-bit start address of the source buffer.

The source end address register contains the end address of the source buffer, used to support circular buffer address mode.

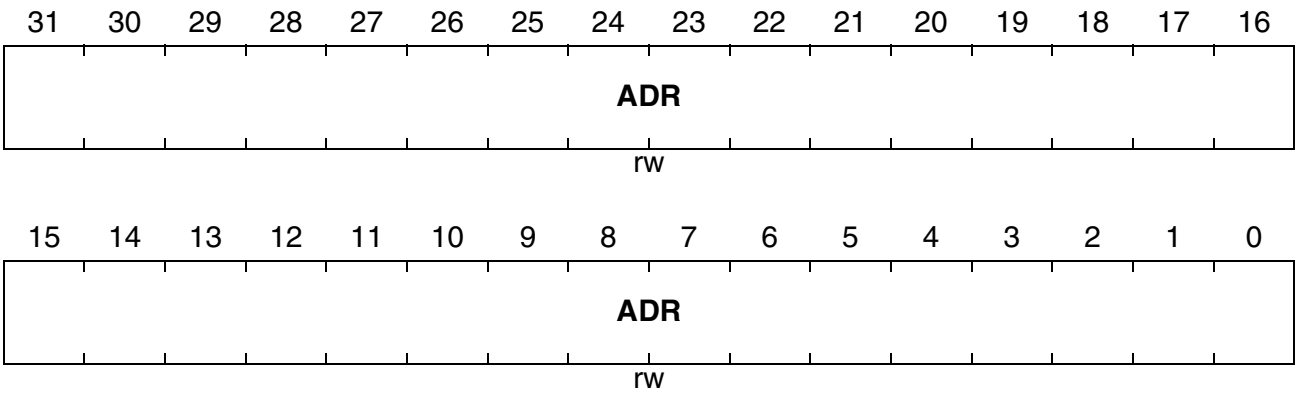
SEAn (n = 00-03 and n = 10-13)
Source End Address Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ADR	[31:0]	rw	**Source End Address** This bit field specifies the 32-bit end address of the source buffer. The value of the source buffer end address is the last address of the source buffer decremented by the source offset.

The destination start address register contains the start address of the destination buffer.

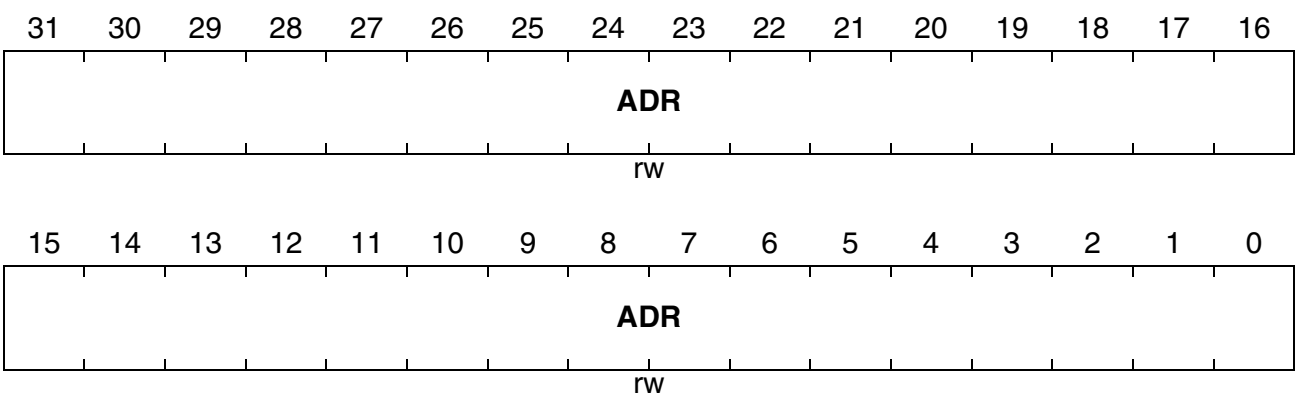
DSAn (n = 00-03 and n = 10-13)
Destination Start Address Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ADR	[31:0]	rw	**Destination Start Address** This bit field specifies the 32-bit start address of the destination buffer.

The destination end address register contains the end address of the destination buffer, used to support circular buffer address mode.

DEAn (n = 00-03 and n = 10-13)
Destination End Address Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ADR	[31:0]	rw	**Destination End Address** This bit field specifies the 32-bit end address of the destination buffer. The value of the end address is the last address of the destination buffer decremented by the destination offset

10.3 DMA Module Implementation

This section describes DMA module interfaces with the clock control, interrupt control, and address decoding.

10.3.1 Interfaces of the DMA Module

Figure 10-13 shows the TC1765 specific implementation details and interconnections of the DMA module. The DMA module is further supplied by a separate clock control, address decoding, interrupt control, port control logic.

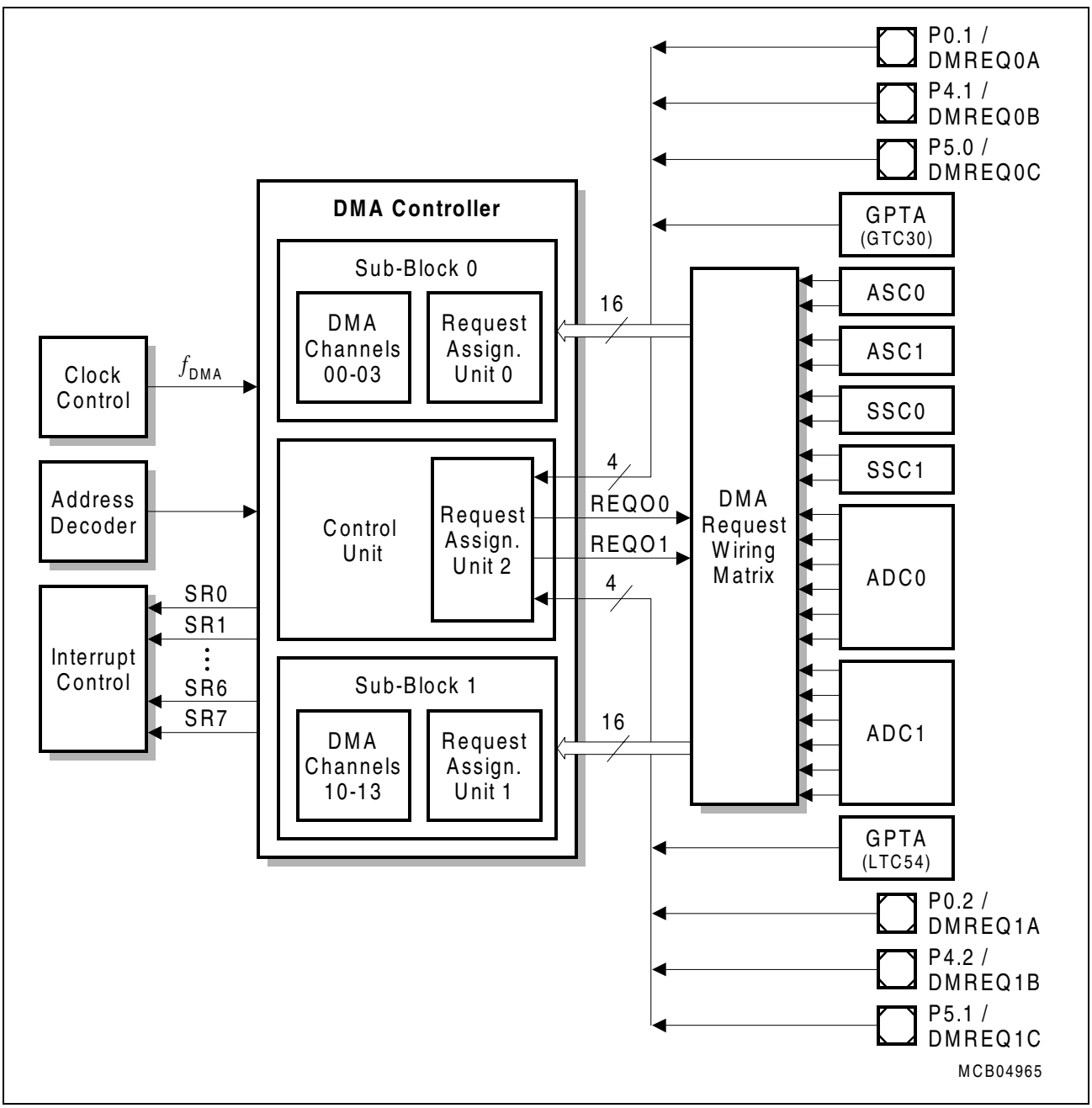


Figure 10-13 DMA Module Implementation and Interconnections

10.3.1.1 Request Assignment Unit 0/1 Input Connections

The DMA request input lines DMAREQ[31:0] are connected with the TC1765 to the DMA request output request lines of the peripheral modules according Table 10-4.

Table 10-4 Request Assignment Units 0/1Input Connections

DMA Request Input	DMA Request Source	DMA Channel	DMA Channel Priority	DMA Block (FPI Bus) Priority
DMA Block 0				
DMAREQ0	SSC0_RDR	DMA Channel 00	high	low
DMAREQ1	SSC0_TDR			
DMAREQ2	ADC0_CH3DR			
DMAREQ3	ADC1_CH14DR			
DMAREQ4	SSC1_RDR	DMA Channel 01	medium high	
DMAREQ5	SSC1_TDR			
DMAREQ6	ADC0_CH5DR			
DMAREQ7	ADC1_CH12DR			
DMAREQ8	SSC1_TDR	DMA Channel 02	medium low	
DMAREQ9	ADC1_CH14			
DMAREQ10	ADC0_CH15DR			
DMAREQ11	ADC1_CH9DR			
DMAREQ12	REQO0	DMA Channel 03	low	
DMAREQ13	REQO1			
DMAREQ14	ASC1_RDR			
DMAREQ15	ASC1_TDR			

Table 10-4 Request Assignment Units 0/1Input Connections (cont'd)

DMA Request Input	DMA Request Source	DMA Channel	DMA Channel Priority	DMA Block (FPI Bus) Priority
DMA Block 0				
DMAREQ16	SSC1_RDR	DMA Channel 10	high	high
DMAREQ17	EXT_R0			
DMAREQ18	ADC0_CH4DR			
DMAREQ19	ADC1_CH13DR			
DMAREQ20	SSC0_TDR	DMA Channel 11	medium high	
DMAREQ21	EXT_R1			
DMAREQ22	ADC0_CH6DR			
DMAREQ23	ADC1_CH11DR			
DMAREQ24	SSC0_TDR	DMA Channel 12	medium low	
DMAREQ25	ADC1_CH13DR			
DMAREQ26	ADC0_CH14			
DMAREQ27	ADC1_CH8			
DMAREQ28	ASC0_RDR	DMA Channel 13	low	
DMAREQ29	ASC0_TDR			
DMAREQ30	REQO0			
DMAREQ31	REQO1			

10.3.1.2 Request Assignment Unit 2 Input Connections

The inputs of the DMA Request Unit 2 are connected to I/O lines according **Table 10-5**

Table 10-5 Request Assignment Unit 2 Input Connections

Input	Port Line	Input	Port Line
REQI0	P0.1 / DMAREQ0A	REQI4	P0.2 / DMAREQ1A
REQI1	P4.1 / DMAREQ0B	REQI5	P4.2 / DMAREQ1B
REQI2	P5.0 / DMAREQ0C	REQI6	P5.1 / DMAREQ1C
REQI3	GTC30	REQI7	LTC54

10.3.2 DMA Module Related External Registers

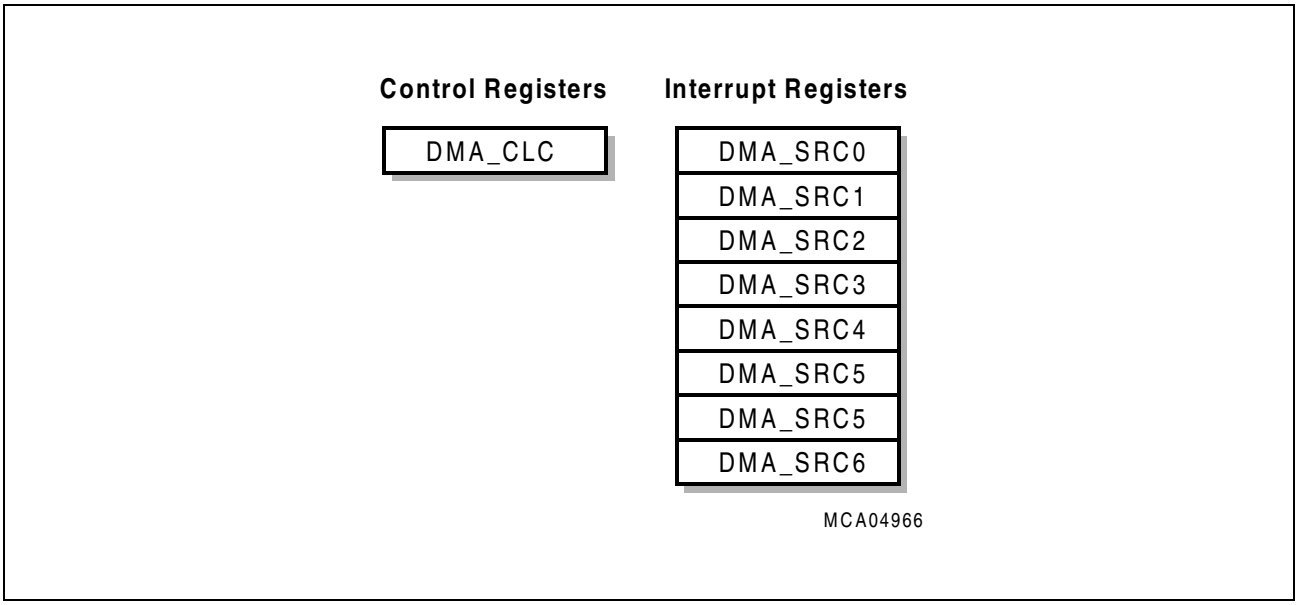
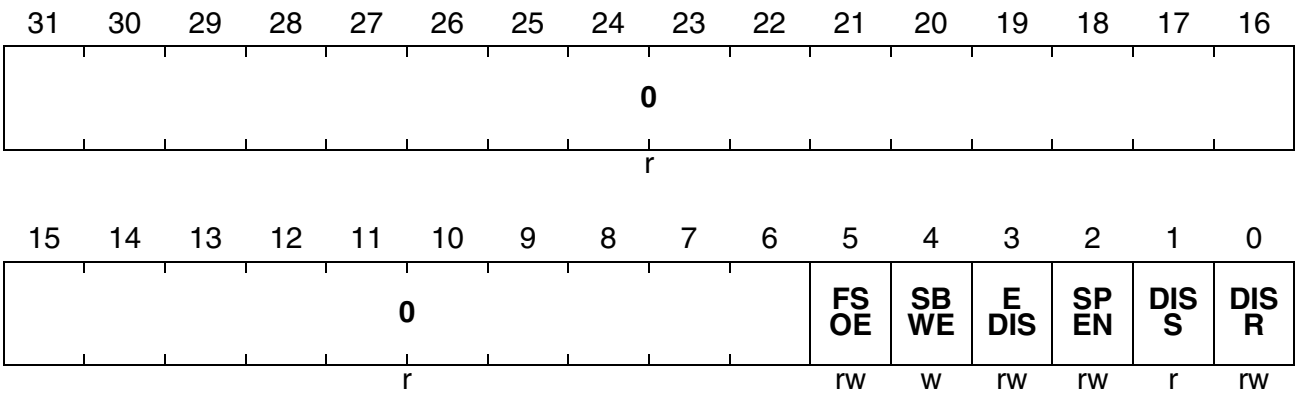


Figure 10-14 DMA Implementation Specific Special Function Registers

10.3.2.1 Clock Control Register

The clock control register allows the programmer to adapt the functionality and power consumption of the DMA module to the requirements of the application. The table below shows the clock control register functionality which is implemented for the DMA module. DMA_CLC is controlling the f_{DMA} clock signal.

DMA_CLC
DMA Clock Control Register **Reset Value: 0000 0003$_H$**



Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module.
SPEN	2	rw	**Module Suspend Enable for OCDS** Used for enabling the suspend mode.
EDIS	3	rw	**External Request Disable** Used for controlling the external clock disable request.
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Defines whether SPEN and FSOE are write protected.
FSOE	5	rw	**Fast Switch Off Enable** Used for fast clock switch off in OCDS suspend mode.
0	[31:6]	r	**Reserved**; returns 0 if read; should be written with 0.

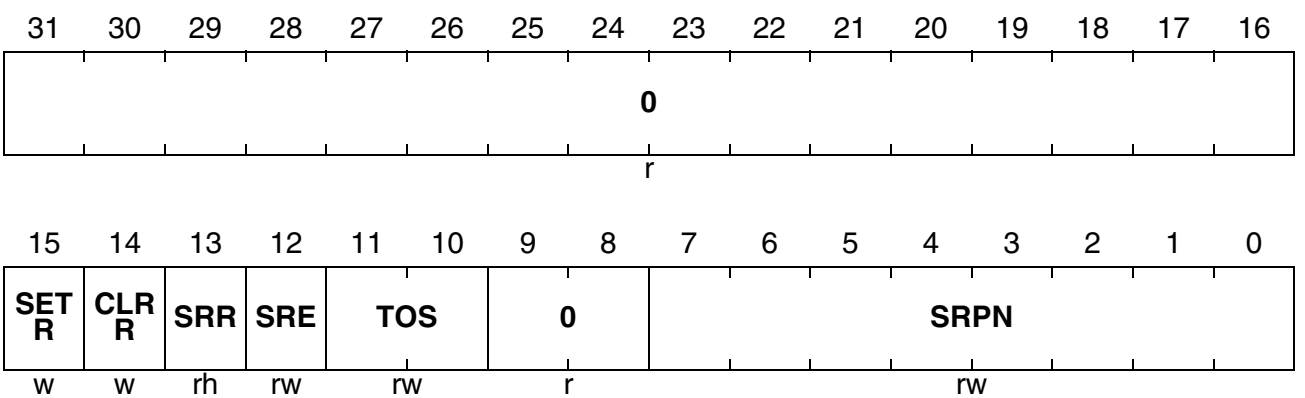
Note: After a hardware reset operation, the DMA module is disabled.

10.3.2.2 Interrupt Registers

The interrupts of the DMA module are controlled by the following service request control registers:

DMA_SRC0
DMA Service Request Control Register 0
DMA_SRC1
DMA Service Request Control Register 1
DMA_SRC2
DMA Service Request Control Register 2
DMA_SRC3
DMA Service Request Control Register 3
DMA_SRC4
DMA Service Request Control Register 4
DMA_SRC5
DMA Service Request Control Register 5
DMA_SRC6
DMA Service Request Control Register 6
DMA_SRC7
DMA Service Request Control Register 7

Reset Values: 0000 0000$_H$



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	[11:10]	rw	**Type of Service Control;** must be written with 00_B.
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**

Field	Bits	Type	Description
0	[9:8], [31:16]	r	**Reserved**; returns 0 if read; should be written with 0.

*Note: Further details on interrupt handling and processing are described in **Chapter 14** of this TC1765 System Units User's Manual.*

10.3.3 DMA Register Address Ranges

In the TC1765, the registers of the DMA controller is located in the following address range:

– DMA controller: Module Base Address = F000 $3C00_H$
Module End Address = F000 $3DFF_H$

Absolute Register Address = Module Base Address + Offset Address
(offset addresses see **Table 10-2**)

11 Memory Protection System

This chapter describes memory protection for the TC1765. Topics covered include the architecture of the memory protection system and the memory protection registers.

11.1 Memory Protection Overview

The TC1765 memory protection system specifies the addressable range and read/write permissions of memory segments available to the currently executing task. The memory protection system controls the position and range of addressable segments in memory. It also controls the kinds of read and write operations allowed within addressable memory segments. Any illegal memory access is detected by the memory protection hardware, which then invokes the appropriate Trap Service Routine (TSR) to handle the error. Thus, the memory protection system protects critical system functions against both software and hardware errors. The memory protection hardware can also generate signals to the Debug Unit to facilitate tracing illegal memory accesses.

As shown in **Figure 11-1**, there are two Memory Protection Register Sets in the TC1765, numbered 0 and 1, which specify memory protection ranges and permissions for code and data. The PSW.PRS bit field determines which of these is the set currently in use by the CPU. Because the TC1765 uses a Harvard-style memory architecture, each Memory Protection Register Set is broken down into a Data Protection Register Set and a Code Protection Register Set. Each Data Protection Register Set can specify up to four address ranges to receive particular protection modes. Each Code Protection Register Set can specify up to two address ranges to receive particular protection modes.

Each of the Data Protection Register Sets and Code Protection Register Sets determines the range and protection modes for a separate memory area. Each contains register pairs which determine the address range (the Data Segment Protection Registers and Code Segment Protection Registers) and one register (Data Protection Mode Register) which determines the memory access modes which apply to the specified range.

The pairs of memory range registers determine the lower address boundary and the upper address boundary of each memory range. The Data Protection Mode Registers and Code Protection Mode Registers determine the access permissions for the ranges specified in their corresponding address range registers.

The memory protection system can also be used to generate signals to the Debug Unit when the processor attempts to access certain memory addresses. When used this way, values in the memory range registers are regarded as individual addresses, instead of defining an address range. An equality comparison with the contents of the address register pairs is performed instead of the normal address range calculation. If enabled for this function, signals are generated to the Debug Unit if the address of a memory access equals any of the address range registers.

Note that while the TriCore architecture allows as many as four Memory Protection Register Sets, the TC1765 implements two; and while the TriCore architecture allows as many as four Code Segment Protection Register Sets, the TC1765 implements two.

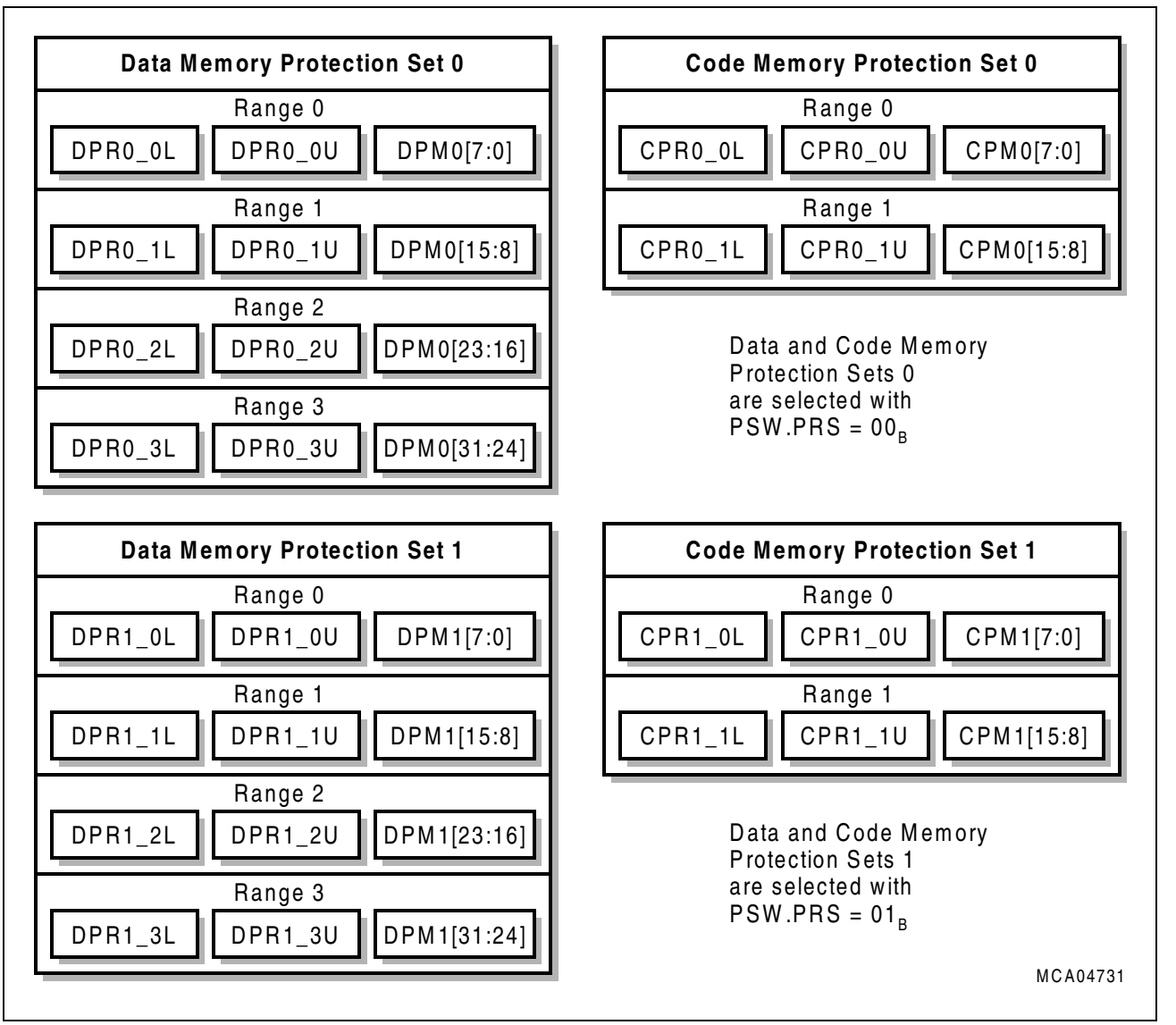


Figure 11-1 Memory Protection Register Sets

11.2 Memory Protection Registers

The TC1765 memory protection architecture is based on memory segments which are specified by address ranges and their associated access permissions or modes. Specific access permissions are associated with each addressable range. Ranges and their associated permissions are specified in two Memory Protection Register Sets (PRS) residing in the Core Special Function Registers (CSFR). A PRS consists of Data Segment Protection Registers, Data Protection Mode Registers, Code Segment Protection Registers, and Code Protection Mode Registers. The organization of these registers is shown in **Figure 11-1**. The PSW_PRS bit field indexes the current PRS. The current PRS determines what accesses can be performed by the processor for each memory segment.

Because of the Harvard-style architecture of the TC1765, each PRS contains separate registers for checking data accesses and code accesses. Memory ranges are specified by pairs of registers which give lower and upper boundary for the associated ranges.

Data and code memory range registers are collectively named DPR*x*_*n*{L,U} and CPR*x*_*n*{L,U}, respectively. In all cases, *x* refers to the specific Memory Protection Register Set that the register is in, *n* refers to the range within the set, and *L* and *U* refer to the lower and upper boundary, respectively. For some lower boundary *L*, upper boundary *U*, and address *a*, the range defined by each address-range register pair is the interval: $L \leq a < U$.

The memory protection system can also be used to generate signals to the Debug Unit when the processor attempts to access particular memory addresses. When used this way, values in the DPR*x*_*n*{L,U} and CPR*x*_*n*{L,U} registers are regarded as individual addresses, instead of defining an address range. An equality comparison with the contents of the address register pairs is performed instead of the normal address range calculation. If enabled for this function, signals are generated to the Debug Unit if the address of a memory access equals any of the DPR*x*_*n*{L,U} and CPR*x*_*n*{L,U} registers.

When used for normal memory protection (not for debugging), the memory protection system performs as outlined in the following paragraphs. When the CPU performs load and store operations, data addresses are checked against the memory ranges given by the current data protection registers. Likewise, when the CPU fetches instructions, the address of the instruction is checked against the memory ranges given by the current code protection registers.

Range checking is disabled if the lower address is greater than the upper address. If the lower address is equal to the upper address, the segment is regarded as empty. If the address does not correspond to an allowable address range in any segment of the current PRS, a trap signal is generated by the memory protection hardware. Note that range checking is also disabled if the mode of a segment indicates that it is to signal the Debug Unit.)

If the address being examined is found to fall within an enabled, non-empty, and allowable range, the associated mode register is checked for access permissions. If the access mode is not allowed, a trap signal is generated by the memory protection hardware.

Table 11-1 shows all registers of the TC1765 Memory Protection Unit.

Table 11-1 Memory Protection Registers

Register Short Name	Register Long Name	Offset Address	Description see
DPR0_0L	Data Segment Protection Register Set 0, Range 0, Lower	0000_H	Page 11-11
DPR0_0U	Data Segment Protection Register Set 0, Range 0, Upper	0004_H	
DPR0_1L	Data Segment Protection Register Set 0, Range 1, Lower	0008_H	
DPR0_1U	Data Segment Protection Register Set 0, Range 1, Upper	$000C_H$	
DPR0_2L	Data Segment Protection Register Set 0, Range 2, Lower	0010_H	
DPR0_2U	Data Segment Protection Register Set 0, Range 2, Upper	0014_H	
DPR0_3L	Data Segment Protection Register Set 0, Range 3, Lower	0018_H	
DPR0_3U	Data Segment Protection Register Set 0, Range 3, Upper	$001C_H$	

Table 11-1 Memory Protection Registers (cont'd)

Register Short Name	Register Long Name	Offset Address	Description see
DPR1_0L	Data Segment Protection Register Set 1, Range 0, Lower	0400_H	Page 11-11
DPR1_0U	Data Segment Protection Register Set 1, Range 0, Upper	0404_H	
DPR1_1L	Data Segment Protection Register Set 1, Range 1, Lower	0408_H	
DPR1_1U	Data Segment Protection Register Set 1, Range 1, Upper	$040C_H$	
DPR1_2L	Data Segment Protection Register Set 1, Range 2, Lower	0410_H	
DPR1_2U	Data Segment Protection Register Set 1, Range 2, Upper	0414_H	
DPR1_3L	Data Segment Protection Register Set 1, Range 3, Lower	0418_H	
DPR1_3U	Data Segment Protection Register Set 1, Range 3, Upper	$041C_H$	Page 11-11
CPR0_0L	Code Segment Protection Register Set 0, Range 0, Lower	1000_H	Page 11-14
CPR0_0U	Code Segment Protection Register Set 0, Range 0, Upper	1004_H	
CPR0_1L	Code Segment Protection Register Set 0, Range 1, Lower	1008_H	
CPR0_1U	Code Segment Protection Register Set 0, Range 1, Upper	$100C_H$	
CPR1_0L	Code Segment Protection Register Set 1, Range 0, Lower	1400_H	Page 11-14
CPR1_0U	Code Segment Protection Register Set 1, Range 0, Upper	1404_H	
CPR1_1L	Code Segment Protection Register Set 1, Range 1, Lower	1408_H	
CPR1_1U	Code Segment Protection Register Set 1, Range 1, Upper	$140C_H$	
DPM0	Set 0 Data Protection Mode Register, Set 0	2000_H	Page 11-12

Table 11-1 Memory Protection Registers (cont'd)

Register Short Name	Register Long Name	Offset Address	Description see
DPM1	Data Protection Mode Register, Set 1	2080_H	Page 11-12
CPM0	Code Protection Mode Register, Set 0	2200_H	Page 11-15
CPM1	Code Protection Mode Register, Set 1	2280_H	Page 11-15

In the TC1765, the memory protection registers are located in the following address range:

– Module Base Address. FFFF $C000_H$
  Module End Address. FFFF $EFFF_H$
– Absolute Register Address = Module Base Address + Offset Address
  (offset addresses see Table 11-1)

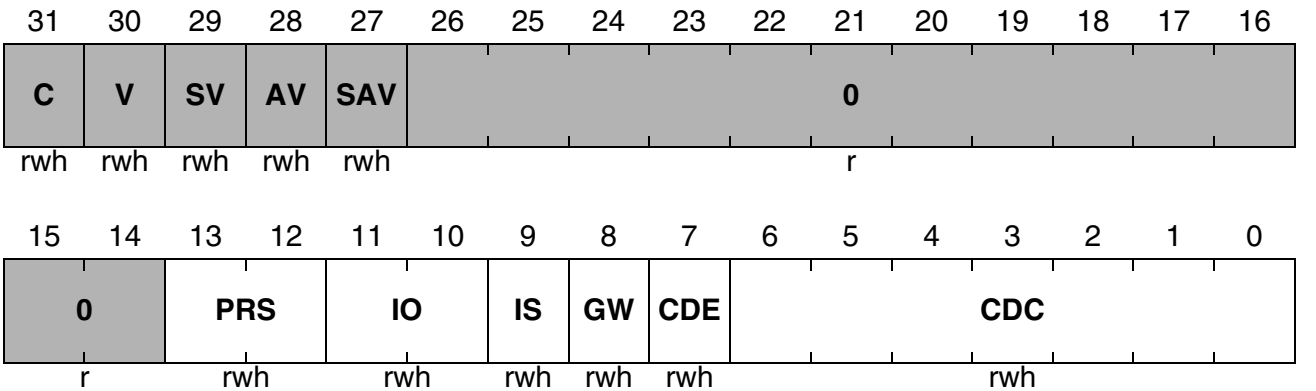
There are two major components within the memory protection system:

– The control bits and bit fields in the PSW.
– The memory protection registers which control program execution and memory access.

11.2.1 PSW Protection Fields

The control fields in the PSW that do not deal with the protection system are shaded in the PSW register table below.

PSW
Program Status Word **Reset Value: 0000 0B80$_H$**


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
C V SV AV SAV 0
rwh rwh rwh rwh rwh r
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 PRS IO IS GW CDE CDC
r rwh rwh rwh rwh rwh rwh

Field	Bits	Type	Description
CDC	[6:0]	rwh	**Call Depth Counter** The CDC field consists of two variable-width fields. The first is a mask field, consisting of a string of zero or more initial 1 bits, terminated by the first 0 bit. The remaining bits of the field are the call depth counter. 0ccccc$_B$ 6-bit counter; trap on overflow 10cccc$_B$ 5-bit counter; trap on overflow 110ccc$_B$ 4-bit counter; trap on overflow 1110cc$_B$ 3-bit counter; trap on overflow 11110c$_B$ 2-bit counter; trap on overflow 111110$_B$ 1-bit counter; trap on overflow 1111110$_B$ Trap every call (call trace mode) 1111111$_B$ Disable call depth counting When the call depth counter overflows, a trap is generated. Depending on the width of the mask field, the call depth counter can be set to overflow at any power of two boundary, from 1 to 64. Setting the mask field to 1111110$_B$ allows no bits for the counter, and causes every call to be trapped. This is used for call tracing. Setting the field to mask field to 1111111$_B$ disables call depth counting altogether.

Field	Bits	Type	Description
CDE	7	rwh	**Call Depth Count Enable** The CDE bit enables call-depth counting, provided that the CDC mask field is not all 1's. CDE is set to 1 by default, but should be cleared by the SYSCALL instruction Trap Service Routine to allow a trapped SYSCALL instruction to execute without producing another trap upon return from the trap handler. It is then set again when the next SYSCALL instruction is executed. 0 Call depth counter disabled 1 Call depth counter enabled
GW	8	rwh	**Global Register Write Permission** GW controls whether the current execution thread has permission to modify the global address registers. Most tasks and ISRs will use the global address registers as "read only" registers, pointing to the global literal pool and key data structures. However, a task or ISR can be designated as the "owner" of a particular global address register, and is allowed to modify it. The system designer must determine which global address variables are used with sufficient frequency and/or in sufficiently time-critical code to justify allocation to a global address register. By compiler convention, global address register A0 is reserved as the base register for short form loads and stores. Register A1 is also reserved for compiler use. Registers A8 and A9 are not used by the compiler, and are available for holding critical system address variables. 0 Write permission to global registers A0, A1, A8, and A9 is disabled 1 Write permission to global registers A0, A1, A8, and A9 is enabled

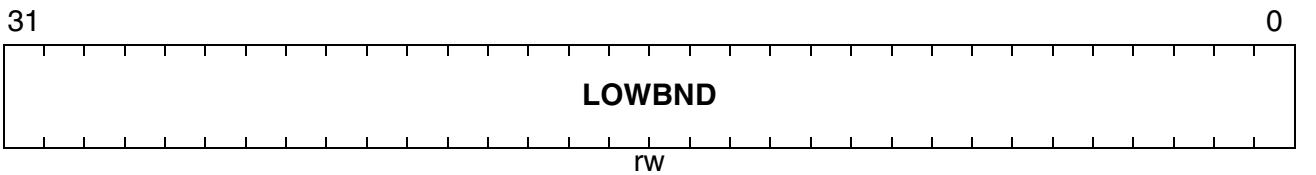
Field	Bits	Type	Description
IS	9	rwh	**Interrupt Stack Control** Determines whether the current execution thread is using the shared global (interrupt) stack or a user stack. 0 U**ser Stack**. If an interrupt is taken when the IS bit is 0, then the stack pointer register is loaded from the ISP register before execution starts at the first instruction of the Interrupt Service Routine. 1 S**hared Global Stack**. If an interrupt is taken when the IS bit is 1, then the current value of the stack pointer register is used by the Interrupt Service Routine.
IO	[11:10]	rwh	**Access Privilege Level Control** This 2-bit field selects determines the access level to special function registers and peripheral devices. 00_B **User-0 Mode**: No peripheral access. Access to segments 14 and 15 is prohibited and will result in a trap. This access level is given to tasks that need not directly access peripheral devices. Tasks at this level do not have permission to enable or disable interrupts. 01_B **User-1 Mode**: regular peripheral access. This access level enables access to common peripheral devices that are not specially protected, including read/write access to serial I/O ports, read access to timers, and access to most I/O status registers. Tasks at this level may disable interrupts. 10_B **Supervisor Mode**. This access level enables access to all peripheral devices. It enables read/write access to core registers and protected peripheral devices. Tasks at this level may disable interrupts. 11_B **Reserved**; this encoding is reserved and is not defined.

Field	Bits	Type	Description
PRS	[13:12]	rwh	**Protection Register Set Selection** The PRS field selects one of two possible sets of memory protection register values controlling load and store operations and instruction fetches within the current process. This field indicates the current protection register set. 00 Protection register set 0 selected 01 Protection register set 1 selected 10 Reserved; don't use this combination 11 Reserved; don't use this combination
0	[26:14]	r	**Reserved**; read as 0; should be written with 0.
–	[31:27]	rwh	Not used for memory protection purposes.

11.2.2 Data Memory Protection Register

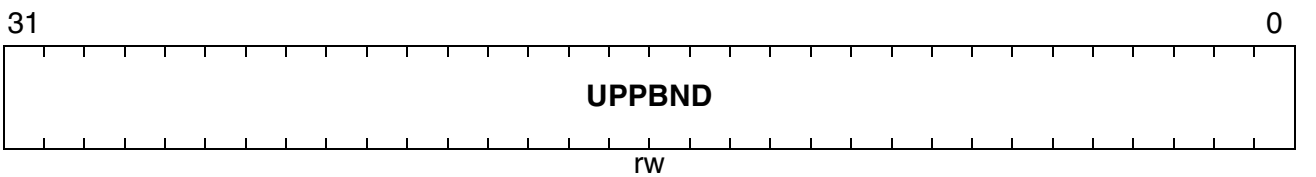
The lower and upper boundaries of a data memory segment are specified by word-length register pairs DPR*x*_*n*L and DPR*x*_*n*U respectively, where *x* is the Memory Protection Register Set number (0..1) and *n* is the range number (0..3).

DPR0_0L DPR0_1L DPR0_2L DPR0_3L
DPR1_0L DPR1_1L DPR1_2L DPR1_3L
Data Segment Protection Register n, Set x, Lower Bound DPRx_nL (x = 0, 1, n = 0-3)
Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
LOWBND	[31:0]	rw	**Lower Boundary Address**

DPR0_0U DPR0_1U DPR0_2U DPR0_3U
DPR1_0U DPR1_1U DPR1_2U DPR1_3U
Data Segment Protection Register n, Set x, Upper Bound DPRx_nU (x = 0, 1, n = 0-3)
Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
UPPBND	[31:0]	rw	**Upper Boundary Address**

The access permissions of the two data memory ranges are specified by the registers DPMx, where *x* is the Memory Protection Register Set number (x = 0, 1). Four byte fields within each DPMx register are assigned to the range number (0..3). Note that in one set the mode register with the four ranges is located in a single word register. Byte field DPMx[7:0] is assigned to Range 0, byte field DPMx[15:8] is assigned to Range 1, byte field DPM[23:16] is assigned to Range 2, and byte field DPMx[31:24] is assigned to Range 3.

DPM0 DPM1

Data Protection Mode Registers DPMx (x = 0, 1) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WE 3	RE 3	WS 3	RS 3	WBL 3	RBL 3	WBU 3	RBU 3	WE 2	RE 2	WS 2	RS 2	WBL 2	RBL 2	WBU 2	RBU 2
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WE 1	RE 1	WS 1	RS 1	WBL 1	RBL 1	WBU 1	RBU 1	WE 0	RE 0	WS 0	RS 0	WBL 0	RBL 0	WBU 0	RBU 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

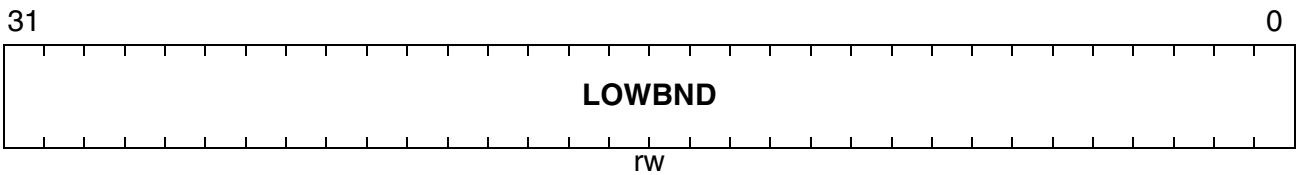
Field	Bits	Type	Description
RBUn (n = 0-3)	0, 8, 16, 24	rw	**Data Read Signal on Upper Bound Access Range n** 0 Data read signal is disabled 1 A signal is asserted to the debug unit on a data read access to an address that matches the upper boundary address of the associated address range.
WBUn (n = 0-3)	1, 9, 17, 25	rw	**Write Signal on Upper Bound Access Range n** 0 Write signal is disabled 1 A signal is asserted to the debug unit on a data write access to an address that matches the upper boundary address of the associated address range.
RBLn (n = 0-3)	2, 10, 18, 26	rw	**Data Read Signal on Lower Bound Access Range n** 0 Data read signal is disabled 1 A signal is asserted to the debug unit on a data read access to an address that matches the lower boundary address of the associated address range.

Field	Bits	Type	Description
WBLn (n = 0-3)	3, 11, 19, 27	rw	**Data Write Signal on Lower Bound Access Range n** 0 Data write signal is disabled 1 A signal is asserted to the debug unit on a data write access to an address that matches the lower boundary address of the associated address range
RSn (n = 0-3)	4, 12, 20, 28	rw	**Address Range Data Read Signal Range n** 0 Data read signal is disabled 1 A signal is asserted to the debug unit on data read accesses to the associated address range
WSn (n = 0-3)	5, 13, 21, 29	rw	**Address Range Data Write Signal Range n** 0 Data write signal is disabled 1 A signal is asserted to the debug unit on data write accesses to the associated address range
REn (n = 0-3)	6, 14, 22, 30	rw	**Address Range Data Read Enable Range n** RE controls reads to the addresses in the associated range. 0 Data read accesses to the associated address range are not permitted 1 Data read accesses to the associated address range are permitted
WEn (n = 0-3)	7, 15, 23, 31	rw	**Address Range Data Write Enable Range n** WE controls writes to the addresses in the associated range. 0 Data write accesses to the associated address range are not permitted 1 Data write accesses to the associated address range are permitted

11.2.3 Code Memory Protection Register

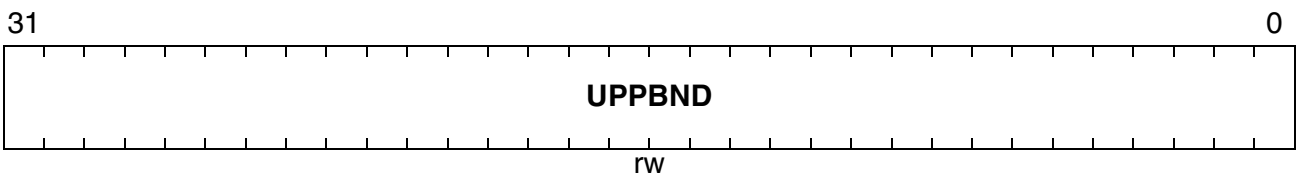
The lower and upper boundaries of a code memory segment are specified by word length register pairs CPR*x*_*n*L and CPR*x*_*n*U respectively, where *x* is the Memory Protection Register Set number (0..1) and *n* is the range number (0..1).

CPR0_0L CPR0_1L
CPR1_0L CPR1_1L
Code Segment Protection Register n, Set x, Lower Bound CPRx_nL (x = 0, 1, n = 0, 1)
Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
LOWBND	[31:0]	rw	**Lower Boundary Address**

CPR0_0U CPR0_1U
CPR1_0U CPR1_1U
Code Segment Protection Register n, Set x, Upper Bound CPRx_nU (x = 0, 1, n = 0, 1)
Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
UPPBND	[31:0]	rw	**Upper Boundary Address**

The access permissions of the two code memory ranges are specified by the registers CPMx, where *x* is the Memory Protection Register Set number (x = 0, 1). Two byte fields within each CPMx register are assigned to the range number (0, 1). Note that in one set, the mode register with the two ranges is located in a single word register. Byte field CPMx[7:0] is assigned to Range 0, and byte field CPMx[15:8] is assigned to Range 1.

CPM0 CPM1

Code Protection Mode Registers CPMx (x = 0, 1) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XE 1	0	XS 1	0	BL 1	0	0	BU 1	XE 0	0	XS 0	0	BL 0	0	0	BU 0
rw	r	rw	r	rw	r	r	rw	rw	r	rw	r	rw	r	r	rw

Field	Bits	Type	Description
BUn (n = 0, 1)	0, 8	rw	**Execute Signal on Upper Bound Access Range n** 0 Upper bound execute signal is disabled 1 A signal is asserted to the debug unit on an instruction fetch to an address that matches the upper bound address of the associated address range
BLn (n = 0, 1)	3, 11	rw	**Execute Signal on Lower Bound Access Range n** 0 Lower bound execute signal is disabled 1 A signal is asserted to the debug unit on an instruction fetch to an address that matches the lower bound address of the associated address range
XSn (n = 0, 1)	5, 13	rw	**Address Range Execute Signal Range n** 0 Execute signal is disabled 1 A signal is asserted to the debug unit on instruction fetches to the associated address range
XEn (n = 0, 1)	7, 15	rw	**Address Range Execute Enable Range n** 0 Instruction fetches to the associated address range are not permitted 1 Instruction fetches to the associated address range are permitted
0	[31:16]	r	**Reserved**; read as 0; should be written with 0.

At any given time, one of the sets is the current protection register set that determines the legality of memory accesses by the current task or ISR. The PRS field in the PSW indicates the current protection register set number. Each protection register set contains separate address range tables for checking data accesses and code accesses. The range table entry is a pair of words specifying a lower and an upper boundary for the associated range. The range defined by one range table entry is the address interval:

- lower bound ≤ address < upper bound

Each range table entry has an associated mode table entry in which access permissions and debug signal conditions for that range are specified. For load and store operations, data address values are checked against the entries in the data range table. For instruction fetches, the PC value for the fetch is checked against the entries in the code range table. When an address is found to fall within a range defined in the appropriate range table, the associated mode table entry is checked for access permissions and debug signal generation.

Modes of Use for Range Table Entries

An individual range table entry can be used for memory protection or for debugging; it is rarely used for both purposes. If the upper and lower bound values have been set for debug breakpoints, they probably are not meaningful for defining protection ranges, and vice versa.However, it is possible — and reasonable — to have some entries in the table for memory protection and others for debugging.

To disable an entry for memory protection, clear both the RE and WE bits in a data range table entry or clear the XE bit in a code range table entry. The entry can be disabled for use in debugging by clearing any debug signal bits. If a range entry is being used for debugging, the debug signal bits that are set determine whether it is used as a single range comparator (giving an in-range/not in-range signal) or as a pair of equal comparators. The two uses are not mutually exclusive.

Using Protection Register Sets

If there were only one protection register set, then either the mappings would need to be general enough to apply to all tasks and ISRs — thus, not terribly useful for isolating software errors in individual tasks — or there would need to be substantial overhead paid on interrupts and task context switches for updating the tables to match the currently executing task or ISR. Those drawbacks are avoided by providing for multiple sets of tables, with two bits in the PSW to select the currently active set.

Note that supervisor mode does not automatically disable memory protection. The protection register set selected for supervisor tasks will normally be set up to allow write access to regions of memory protected from user mode access. In addition, of course, supervisor tasks can execute instructions to change the protection maps, or to disable the protection system entirely. But supervisor mode does not implicitly override memory protection, and it is possible for a supervisor task to take a memory protection trap.

11.3 Sample Protection Register Set

Figure 11-2 illustrates Data Protection Register Set *n*, where *n* is one of the two sets selected by the PSW.PRS field. Each register set in this example consists of four range table entries. The defined ranges can potentially overlap or be nested. Nesting of ranges can be used, for example, to allow write access to a subrange of a larger range in which the current task is allowed read access. The four Data Segment Protection Registers and four Data Protection Mode Registers are set up as follows:

- Data Segment Protection Register 3 (DPRn_3) defines the upper and lower boundaries for Data Range 4. Data Protection Mode Register 3 (DPMn_3) defines the permissions and debug conditions for Data Range 4.
- Data Segment Protection Register 2 (DPRn_2) defines the upper and lower boundaries for Data Range 3. Data Protection Mode Register 2 (DPMn_2) defines the permissions and debug conditions for Data Range 3. Note that Data Range 3 is nested within Data Range 4.
- Data Segment Protection Register 1 (DPRn_1) defines the upper and lower boundaries for Data Range 2. Data Protection Mode Register 1 (DPMn_1) defines the permissions and debug conditions for Data Range 2.
- Data Segment Protection Register 0 (DPRn_0) defines the upper and lower boundaries for Data Range 1. Data Protection Mode Register 0 (DPMn_0) defines the permissions and debug conditions for Data Range 1.

This same configuration can be used to illustrate Code Protection Register Set n.

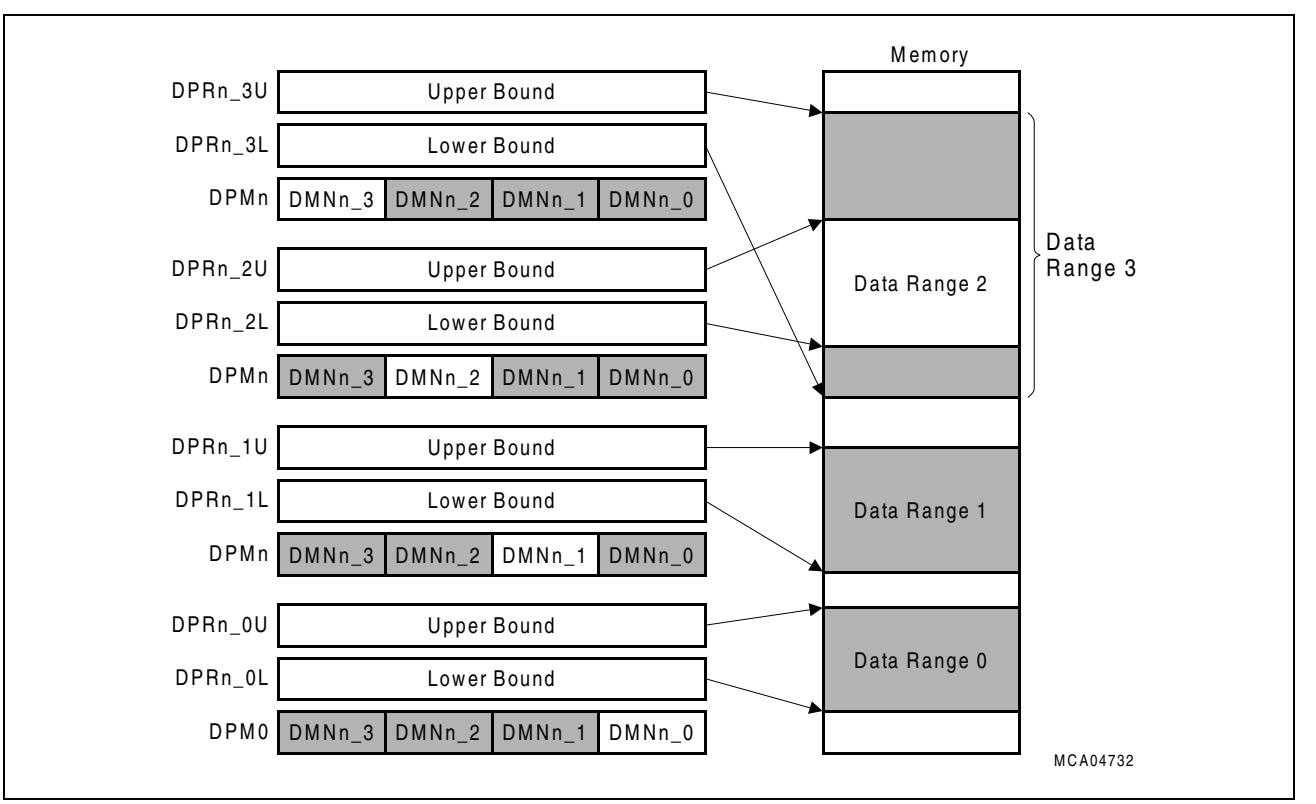


Figure 11-2 Example Configuration of a Data Protection Register Set (Set *n*)

11.4 Memory Access Checking

If the protection system is enabled, before any memory access (read, write, execute) is performed, it is checked for legality as determined by all of the following:

- The protection enable bits in the SYSCON Register,
- The current I/O privilege level (0 = User-0 Mode; 1 = User-1 Mode; 2 = Supervisor Mode), and
- The ranges defined in the currently selected protection register set.

Data addresses (read and write accesses) are checked against the currently selected data address range table, while instruction fetch addresses are checked against the code address range tables. The mode entries for the data range table entries enable only read and write accesses, while the mode entries for the code range table entries enable only execute access. In order for data to be read from program space, there must be an entry in the data address range table that covers the address being read. Conversely, there must be an entry in the code address range table for the instruction being read.

Access to the internal and external peripherals is through the two upper segments of the TC1765 address space (high-order address bits equal to 1110_B and 1111_B). Access checking for addresses in the peripheral segments is independent of access checking in the remainder of the address space. Access to peripheral segments is not allowed for tasks at I/O privilege Level 0 (User-0 tasks). Tasks at I/O privilege Level 1 and higher have access rights to the peripheral segment space. However, the validity of any access attempt depends on the presence of a peripheral at the accessed address, and any restrictions it may impose on its own access. Protected peripherals, for example, require I/O privilege Level 2, as reflected by the supervisor line value on the system bus.

If the memory protection system is disabled, any access to any memory address outside of the peripheral segments is permitted, regardless of the I/O privilege level. There are no memory regions reserved for supervisor access only, when the memory protection system is disabled.

When the memory protection system is enabled, for an access to be permitted, the address for the access must fall within one or more of the ranges specified in the currently selected protection register set. Furthermore, the mode entry for at least one of the matching ranges must enable the requested type of access.

11.4.1 Permitted versus Valid Accesses

A memory access can be permitted within the ranges specified in the data and code range tables without necessarily being valid. A range specified in a range table entry could cover one or more address regions where no physical memory was implemented. Although that would normally reflect an error in the system code that set up the address range, the memory protection system only uses the range table entries when determining whether an access is permitted. In addition, if the memory protection system

is disabled, all accesses must be taken as permitted, although individual accesses may or may not be valid.

An access that is not permitted under the memory protection system results in a memory protection trap. When permitted, an access to an unimplemented memory address results in a bus error trap, provided that the memory address is in one of the segments reserved for local memory. If the address is an external memory address, the result depends on the memory implementation, and is not architecturally defined. An access can also be permitted but invalid due to a misaligned address. Misaligned accesses result in an alignment trap, rather than a protection trap.

11.4.2 Crossing Protection Boundaries

An access can straddle two regions. For example, **Figure 11-3** illustrates the condition where Instruction A lies in an execute region of memory, Instruction C lies in a no-execute region of memory, and Instruction B straddles the execute/no execute boundary.

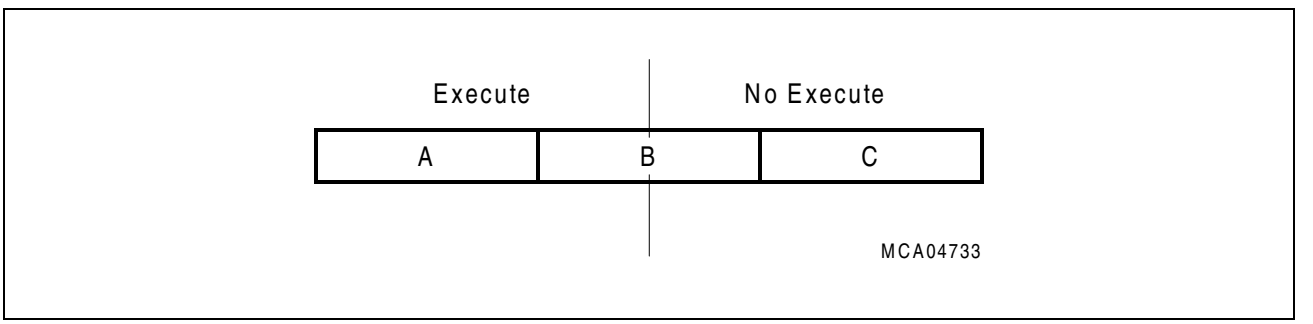


Figure 11-3 Protection Boundaries

Because the PC is used in the comparison with the range registers, the program error exception is not signaled until Instruction C is fetched. The same is true for all comparisons — the address of the first accessed byte is compared against the memory protection range registers. Hence, an access assumes the memory protection properties of the first byte in the access regardless of the number of bytes involved in the access.

For normal accesses, this assumption is not a problem because the regions are set up according to the natural access boundaries for the code or data that the region contains. For wild accesses due to software or hardware errors, stores are the main concern. In the worst case, a double-word store that is aligned on a half-word boundary can extend three half-words beyond the end of the region in which its address lies.

One way to prevent boundary crossings is to leave at least three half-words of buffer space between regions. This configuration prevents wild stores from destroying data in adjacent read-only regions, for example.

12 Parallel Ports and Pins

The TC1765 has 77 digital input/output port lines organized into four parallel 16-bit ports (Port 0 to Port 3), one 8-bit port (Port 4), and one 5-bit port (Port 5). Additionally, 24 analog input port lines are available. The EBU is provided with dedicated data, address, and control lines. A 16-bit Trace Port is available only in the TC1765T.

The digital parallel ports Port 0 to Port 5 can be all used as general purpose I/O lines or they can perform input/output functions for the on-chip peripheral units. The on-chip External Bus Interface Unit allows to communicate with external memories, external peripherals, or external debugging devices. An overview on the port-to-peripheral unit assignment is shown in **Figure 12-2**.

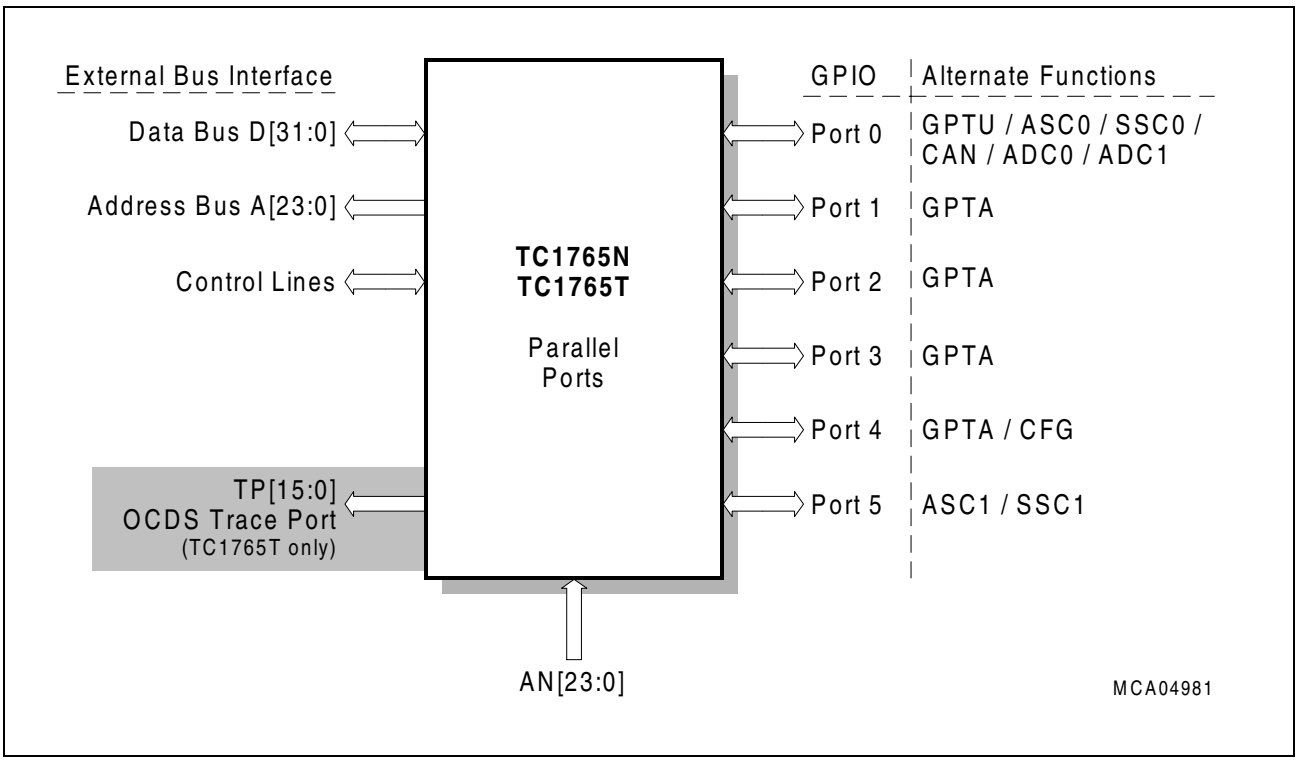


Figure 12-1 Parallel Ports of the TC1765

12.1 General Port Operation

Figure 12-2 shows a general block diagram of an TC1765 port line. Each port line is equipped with a number of control and data bits, enabling very flexible usage of the line.

Each port pin can be configured for input or output operation. In input mode (default after reset), the output driver is switched off (high-impedance). The actual voltage level present at the port pin is translated into a logic 0 or 1 via a Schmitt-Trigger device and can be read via the read only register Px_IN. In output mode, the output driver is activated and drives the value supplied through the multiplexer to the port pin. Switching between input and output mode is accomplished through the Px_DIR register, which

enables or disables the output driver. Alternatively, a peripheral unit can define the port direction (via AltDir) if it uses bidirectional I/O lines.

The output multiplexer in front of the output driver enables the port output function to be used for different purposes. If the pin is used as general purpose output, the multiplexer is switched by software to the Output Data Register Px_OUT. Software can set or clear the bit in Px_OUT, and therefore it can directly influence the state of the port pin. If the on-chip peripheral units use the pin for output signals (line AltEnable active), alternate output lines can be switched via the multiplexer to the output driver circuitry.

Latch Px_IN is provided for input functions of the on-chip peripheral units. Its input is connected to the output of the input Schmitt-Trigger. Further, an input signal can be connected directly to the various inputs of the peripheral units (AltDataIn). The function of the input line from the pin to the input latch Px_IN and to AltDataIn is independent of the port pin operates as input or output. This means that when the port is in output mode, the level of the pin can be read by software via latch Px_IN or a peripheral can use the pin level as an input. This offers additional advantages in an application.

- Each port line can also be programmed to activate an internal weak pull-up or pull-down device. Register Px_PUDSEL selects whether a pull-up or the pull-down device is activated while register Px_PUDEN enables or disables the pull devices.
- The data written to the output register Px_OUT by software can be used as input data to an on-chip peripheral. This enables, for example, peripheral tests via software without external circuitry. Examples for this can be the triggering of a timer count input, generating an external interrupt, or simulating the incoming serial data stream to a serial port receive input via software.
- When the pin is used as an output, the actual logic level at the pin can be examined through reading latch Px_IN and compared against the applied output level (either applied through software via the output register Px_OUT, or via an alternate output function of a peripheral). This can be used to detect some electrical failures at the pin caused through external circuitry. In addition, software supported arbitration schemes can be implemented in this way using the open-drain configuration and an external wired-And circuitry. Collisions on the external communication lines can be detected when a logic 1 is output, but a logic 0 is seen when reading the pin value via the input latch Px_IN.
- The output data from a peripheral applied to the pin via an alternate output function can be read through software or can be used by the same or another peripheral as input data. This enables testing of peripheral functions or provides additional connections between on-chip peripherals via the same pin without external wires.

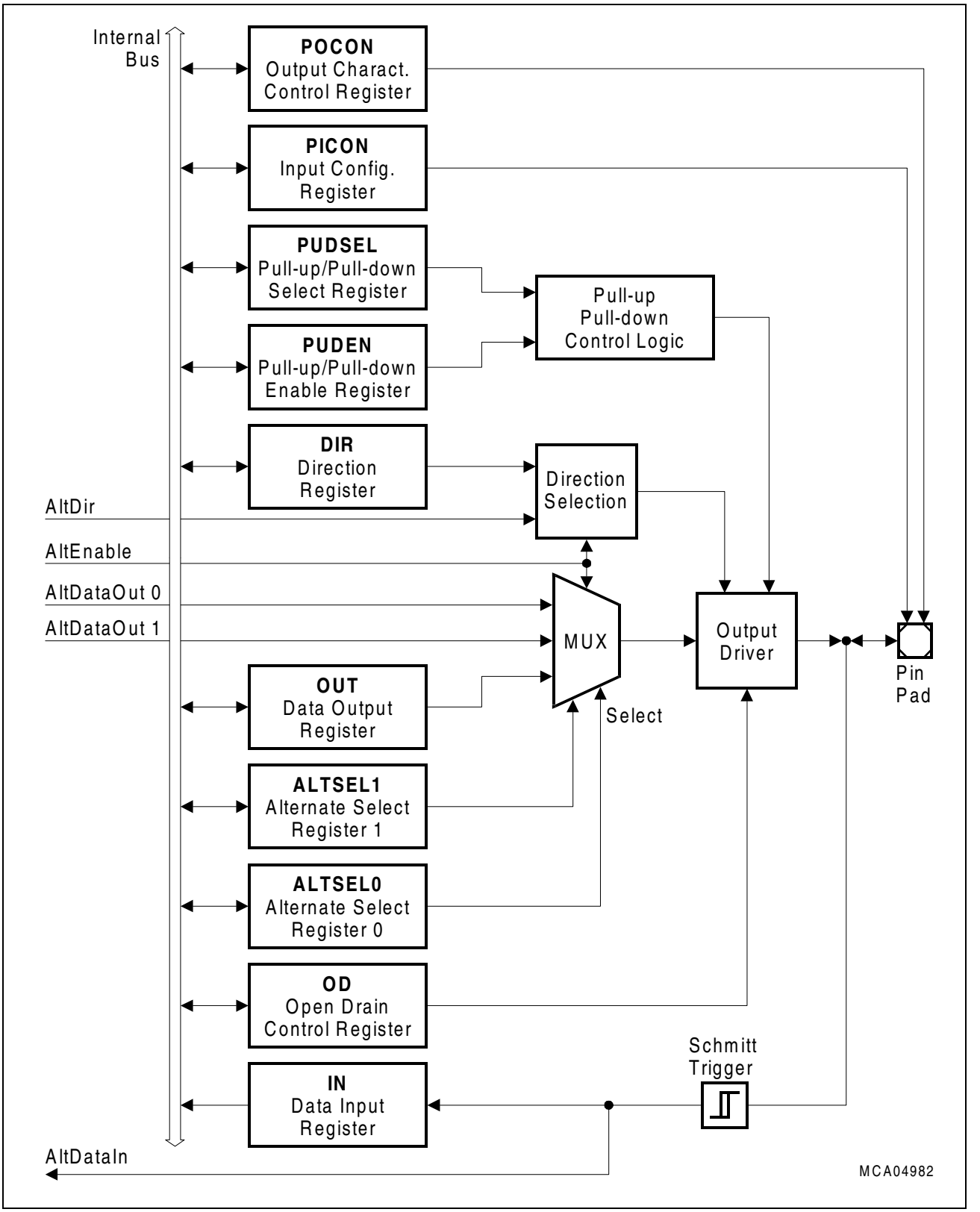


Figure 12-2 General Port Structure

12.2 Port Registers

The individual control and data bits of each digital parallel port are implemented in a number of registers. Bits with the same meaning and function are assembled together in the same register. Each parallel port, except the analog ports P6 and P7, consists of a set of registers. The registers are used to configure and use the port as general purpose I/O or alternate function input/output. For most ports not all registers are implemented. The availability of the registers in the specific ports is defined in Section 12.3 to Section 12.8.

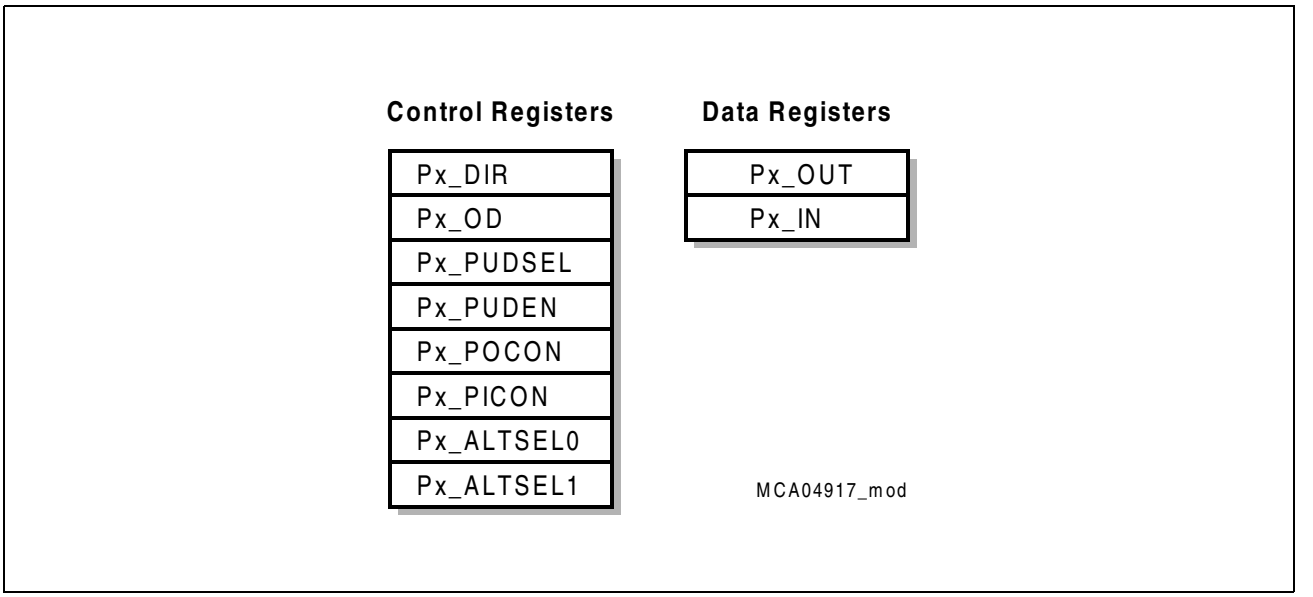


Figure 12-3 Port Registers

Table 12-1 Port Registers

Register Short Name	Register Long Name	Offset Address	Description see
Px_OUT	Port x Data Output Register	0010_H	Page 12-6
Px_IN	Port x Data Input Register	0014_H	Page 12-7
Px_DIR	Port x Direction Control Register	0018_H	Page 12-8
Px_OD	Port x Open Drain Control Register	$001C_H$	Page 12-9
Px_PUDSEL	Port x Pull-up/-down Select Register	0028_H	Page 12-11
Px_PUDEN	Port x Pull-up/-down Enable Register	$002C_H$	Page 12-12
Px_POCON	Port x Output Characteristic Control Register	0030_H	Page 12-13
Px_PICON	Port x Input Configuration Register	0040_H	Page 12-10
Px_ALTSEL0	Port x Alternate Select Register 0	0044_H	Page 12-14
Px_ALTSEL1	Port x Alternate Select Register 1	0048_H	Page 12-14

In the TC1765, the registers of the digital ports are located in the address ranges as shown in **Table 12-2**.

Table 12-2 Port Registers Address Ranges

Port No.	Address Range
Port 0	F000 2800_H - F000 $28FF_H$
Port 1	F000 2900_H - F000 $29FF_H$
Port 2	F000 $2A00_H$ - F000 $2AFF_H$
Port 3	F000 $2B00_H$ - F000 $2BFF_H$
Port 4	F000 $2C00_H$ - F000 $2CFF_H$
Port 5	F000 $2D00_H$ - F000 $2DFF_H$

– Absolute Register Address = Module Base Address (**Table 12-2**) + Offset Address (**Table 12-1**)

12.2.1 Data Output Register

If a port pin is used as general purpose output (GPIO), output data is written into register Px_OUT of port x.

Px_OUT
Port x Data Output Register **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	**P14**	**P13**	**P12**	**P11**	**P10**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
Pn **(n = 15-0)**	n	rw	**Port x Pin n Output Value** 0 Port x pin n output value = 0 (default after reset) 1 Port x pin n output value = 1
0	[31:16]	r	**Reserved**; read as 0; should be written with 0;

The contents of Px_OUT.n are output on the assigned pin if the pin is assigned as GPIO pin and the direction is switched/set to output (Px_DIR.n = 1). A read operation of Px_OUT returns the register value and not the state of the Px pins.

12.2.2 Data Input Register

The value at a port pin can be read through the read-only register Px_IN. The data input register Px_IN always contains a latched value of the assigned port pin.

Px_IN
Port x Data Input Register **Reset Value: 0000 XXXX$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
Pn **(n = 15-0)**	n	r	**Port x Pin n Latched Input Value** 0 Port x input pin n latched value = 0 1 Port x input pin n latched value = 1
0	[31:16]	r	**Reserved**; read as 0; should be written with 0;

12.2.3 Direction Control Register

The direction of port pins can be controlled in the following ways:

- Always controlled by Px_DIR register
- Controlled by Px_DIR register if used for GPIO and controlled by the peripheral if used for alternate function
- Controlled by Px_DIR register if used as GPIO and fixed direction if used for alternate function
- Always fixed if used for GPIO and alternate function

If the port direction is controlled by the respective direction register Px_DIR, the following encoding is defined:

Px_DIR
Port x Direction Register **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	**P14**	**P13**	**P12**	**P11**	**P10**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

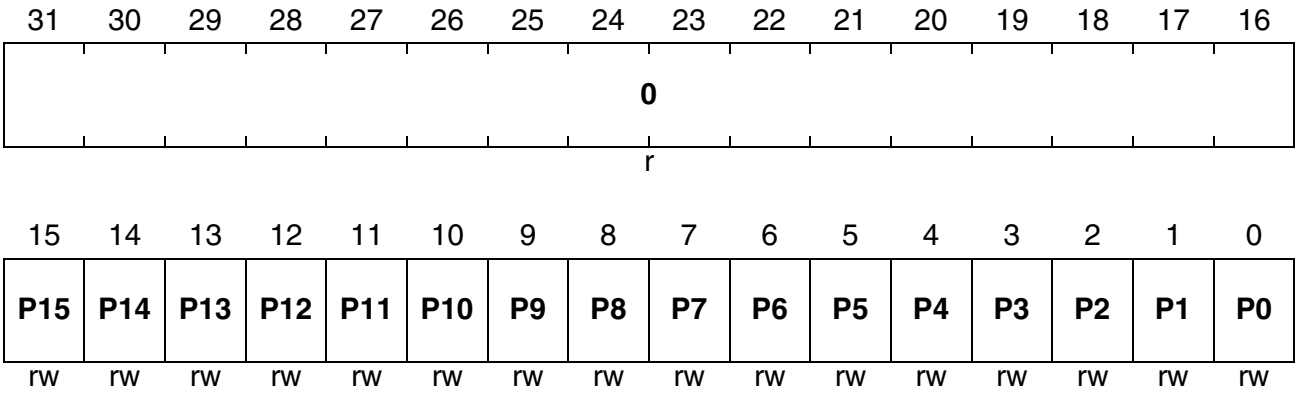
Field	Bits	Type	Description
Pn **(n = 15-0)**	n	rw	**Px Pin n Direction Control** 0 Direction is set to input (default after reset) 1 Direction is set to output
0	[31:16]	r	**Reserved;** read as 0; should be written with 0;

12.2.4 Port Open Drain Control Register

For ports P0 to P5, each pin in output mode can be switched to Open Drain Mode. If driven with 1, no driver will be activated; if driven with 0, the pull-down transistor will be activated.

The open drain mode is controlled by the register **Px_OD**.

Px_OD
Port x Open Drain Control Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
Pn **(n = 15-0)**	n	rw	**Port x Pin n Open Drain Mode** 0 Normal Mode, output is actively driven for 0 and 1 state. 1 Open Drain Mode, output is actively driven only for 0 state.
0	[31:16]	r	**Reserved**; read as 0; should be written with 0;

12.2.5 Input Configuration Register

The input threshold of ports P0 to P5 can be selected pin-wise to be TTL or CMOS-like via the related Px_PICON registers.

Px_PICON
Port x Input Configuration Register **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	**P14**	**P13**	**P12**	**P11**	**P10**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
Pn **(n = 15-0)**	n	rw	**Port x Pin n Input Threshold Type** 0 Pin n of port x has TTL input threshold 1 Pin n of port x has CMOS-like input threshold
0	[31:16]	r	**Reserved**; read as 0; should be written with 0;

12.2.6 Pull-Up/-Down Device Control

Internal pull-up/pull-down devices can be optionally applied to a port pin. This offers the possibility to configure the following input characteristics:

- tri-state
- high-impedance with a weak pull-up device
- high-impedance with a weak pull-down device

and the following output characteristics:

- push/pull (optional pull-up/pull-down)
- open drain with internal pull-up
- open drain with external pull-up

The pull-up/pull-down device can be fixed or controlled via the registers **Px_PUDSEL** and **Px_PUDEN**. Register **Px_PUDSEL** selects the type of pull-up/pull-down device, while register **Px_PUDEN** enables or disables it. The pull-up/pull-down device can be selected pin-wise. Note that the pull-up/pull-down devices are predefined for some pins after reset (see port P3 and P4).

Px_PUDSEL
Port x Pull-Up/Pull-Down Select Register **Reset Value: (dependant on port)**

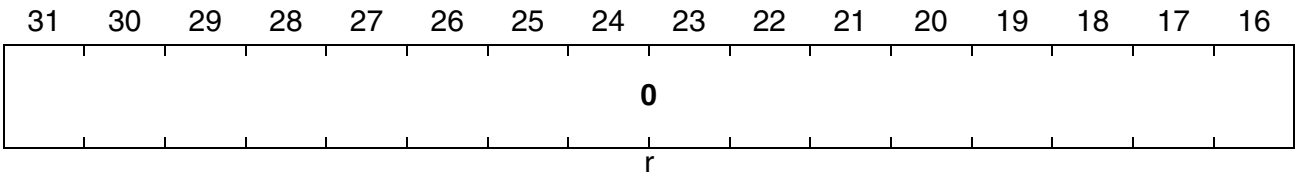
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

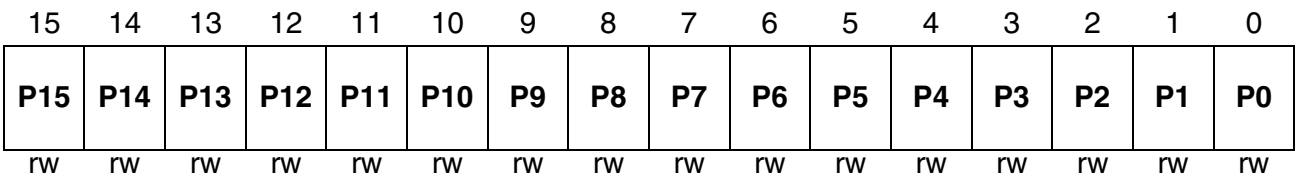
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	**P14**	**P13**	**P12**	**P11**	**P10**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
Pn **(n = 15-0)**	n	rw	**Pull-Up/-Down Device Select Port x Bit n** 0 Pull-down device is selected 1 Pull-up device is selected
0	[31:16]	r	**Reserved**; read as 0; should be written with 0;

Note: The selected pull-up/pull-down device is enabled by setting the respective bit in the Px_PUDEN register.

Px_PUDEN
Port x Pull-Up/-Down Enable Register **Reset Value: (dependent on port)**





Field	Bits	Type	Description
Pn **(n = 15-0)**	n	rw	**Pull-Up/Pull-Down Device Enable at Port x Bit n** 0 Pull-up or pull-down device is disabled 1 Pull-up or pull-down device is enabled
0	[31:16]	r	**Reserved**; read as 0; should be written with 0;

Note: Pull-ups/pull-downs can also be activated during power-down by setting bit SCU_CON.DISPRDPD to 0 (default after reset).

12.2.7 Output Characteristics Control Register

For a significantly improvement of the overall EMC (Electromagnetic Compatibility) behavior of the TC1765, the output driver characteristics of digital ports can be configured by software.

Two type of characteristics can be selected:

- **Edge Characteristic** defines the rise/fall time for the respective port, that is, the transition time. Slow edges reduce the peak currents that are drawn when changing the voltage level of an external capacitive load.
- **Driver Characteristic** defines either the general driving capability of the respective driver (high or low current mode). Reducing the driver strength increases the output's internal resistance which attenuates noise that is imported via the output line.

The output characteristics are pin-wise controlled for ports P0 to P5 using register **Px_PCON**.

Px_POCON
Port x Output Driver Characteristic Control Register **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Px PEC 15	Px PDC 15	Px PEC 14	Px PDC 14	Px PEC 13	Px PDC 13	Px PEC 12	Px PDC 12	Px PEC 11	Px PDC 11	Px PEC 10	Px PDC 10	Px PEC 9	Px PDC 9	Px PEC 8	Px PDC 8
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Px PEC 7	Px PDC 7	Px PEC 6	Px PDC 6	Px PEC 5	Px PDC 5	Px PEC 4	Px PDC 4	Px PEC 3	Px PDC 3	Px PEC 2	Px PDC 2	Px PEC 1	Px PDC 1	Px PEC 0	Px PDC 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
PDCn (n = 15-0)[1)]	see register diagrams above	rw	**Driver Characteristic Control of Port x Pin n** 0 High Current Mode; driver operates with maximum strength. 1 Low Current Mode; driver always operates with reduced strength.
PECn (n = 15-0)[1)]	see register diagrams above	rw	**Edge Characteristic Control of Port x Pin n** 0 Normal timing 1 Slow timing
0	[31:16]	r	**Reserved**; read as 0; should be written with 0.

1) For Port 4 only P4_POCON[15:0] are available; for Port 5 only P4_POCON[9:0] are available;

12.2.8 Alternate Port Functions

Most of the port lines are assigned to certain programmable alternate input or output functions. Alternate functions can be:

- External DMA request inputs
- Input/output functions of timers
- Input/output functions of serial interfaces
- A/D Converter control lines

12.2.8.1 Alternate Input Functions

The number of alternate functions that uses a pin for input is not limited. Each port control logic of an I/O pin provides several input paths:

- Digital input value via register
- Direct digital input value

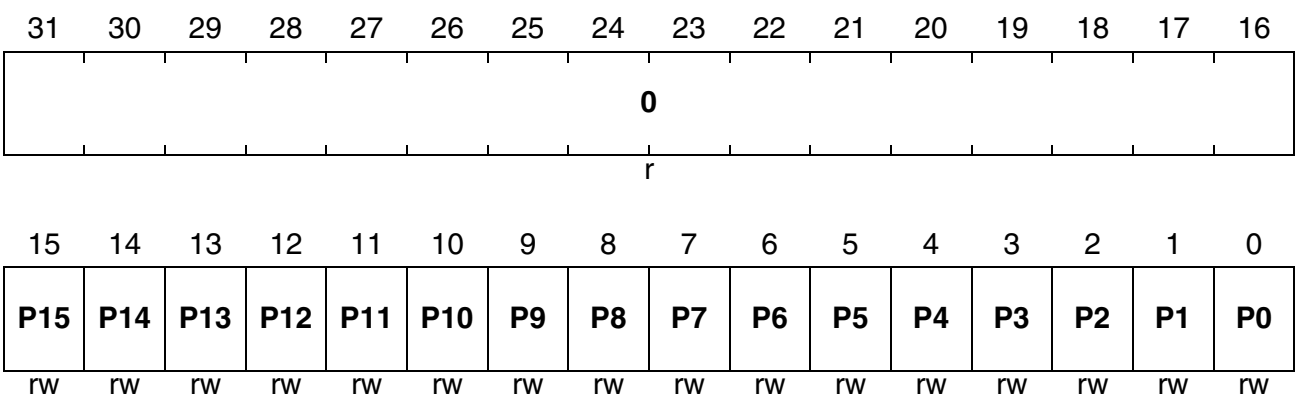
12.2.8.2 Alternate Output Functions

Alternate functions are selected via an output multiplexer which can select up to four output lines. This multiplexer can be controlled by the following three signals:

- Register Px_ALTSEL0
- Register Px_ALTSEL1
- Signal AltEnable

Selection of alternate functions are defined in registers Px_ALTSEL0 and Px_ALTSEL1. The tables in the port sections **Section 12.3** to **Section 12.8** define which type of select signal is used for the alternate function selection for each port pin.

Px_ALTSELn (n = 1,0)
Port x Alternate Function Select Register **Reset Value: 0000 0000$_H$**



12.3 Port 0

Port 0 is a general purpose 16-bit bidirectional I/O port. It serves as GPIO lines or as alternate function I/O lines for the following peripheral modules:

- Digital I/O lines of the Analog/Digital Converters ADC0 and ADC1
- I/O lines of the GPTU
- I/O lines of the Asynchronous/Synchronous Serial Interface ASC0
- I/O lines of the Synchronous Serial Interface SSC0
- I/O lines of the TwinCAN Module
- DMA request inputs

A port line of Port 0 can be used as standard GPIO pins if its alternate function is not required.

12.3.1 Features

- Push/pull output drivers
- Optional Open Drain Output Mode (programmable via register P0_OD)
- 5.0 Volt operation for GPIO
- Pin-wise programmable input threshold (TTL or CMOS-like) via register P0_PICON
- Programmable slew-rate and output driver strength via register P0_POCON
- Programmable pull-up/pull-down devices via registers P0_PUDSEL and P0_PUDEN

12.3.2 Registers

The following port registers are available at Port 0:

Table 12-3 Port 0 Registers

Register Short Name	Register Long Name
P0_OUT	Port 0 Data Output Register
P0_IN	Port 0 Data Input Register
P0_DIR	Port 0 Direction Register
P0_OD	Port 0 Open Drain Mode Register
P0_PUDSEL	Port 0 Pull-up/Pull-down Select Register
P0_PUDEN	Port 0 Pull-up/Pull-down Enable Register
P0_POCON	Port 0 Port Output Characteristics Control Register
P0_PICON	Port 0 Input Configuration Register
P0_ALTSEL0	Port 0 Alternate Select Register 0
P0_ALTSEL1	Port 0 Alternate Select Register 1

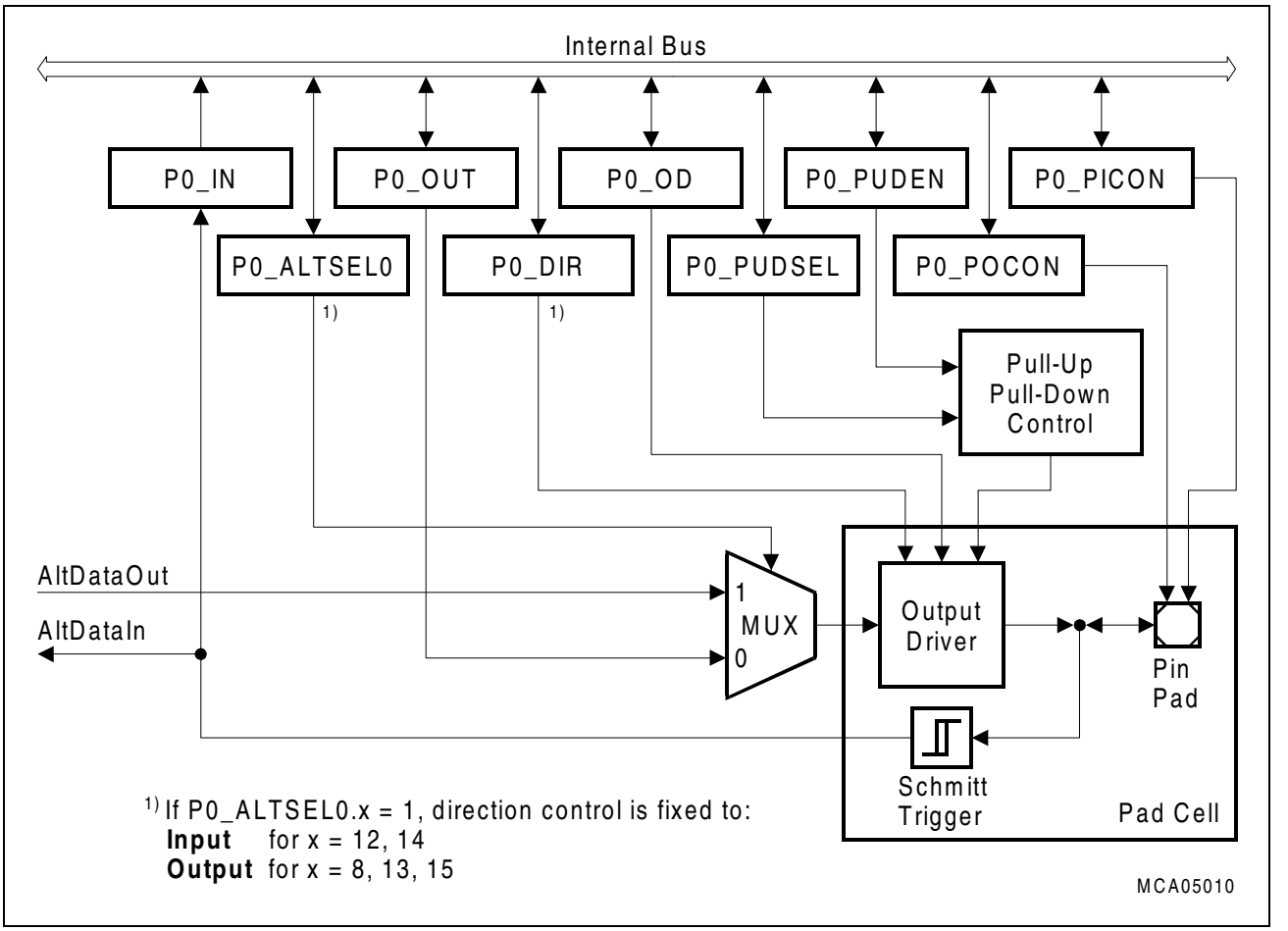


Figure 12-4 Port 0 Configuration for P0[3:0] and P0[15:7]

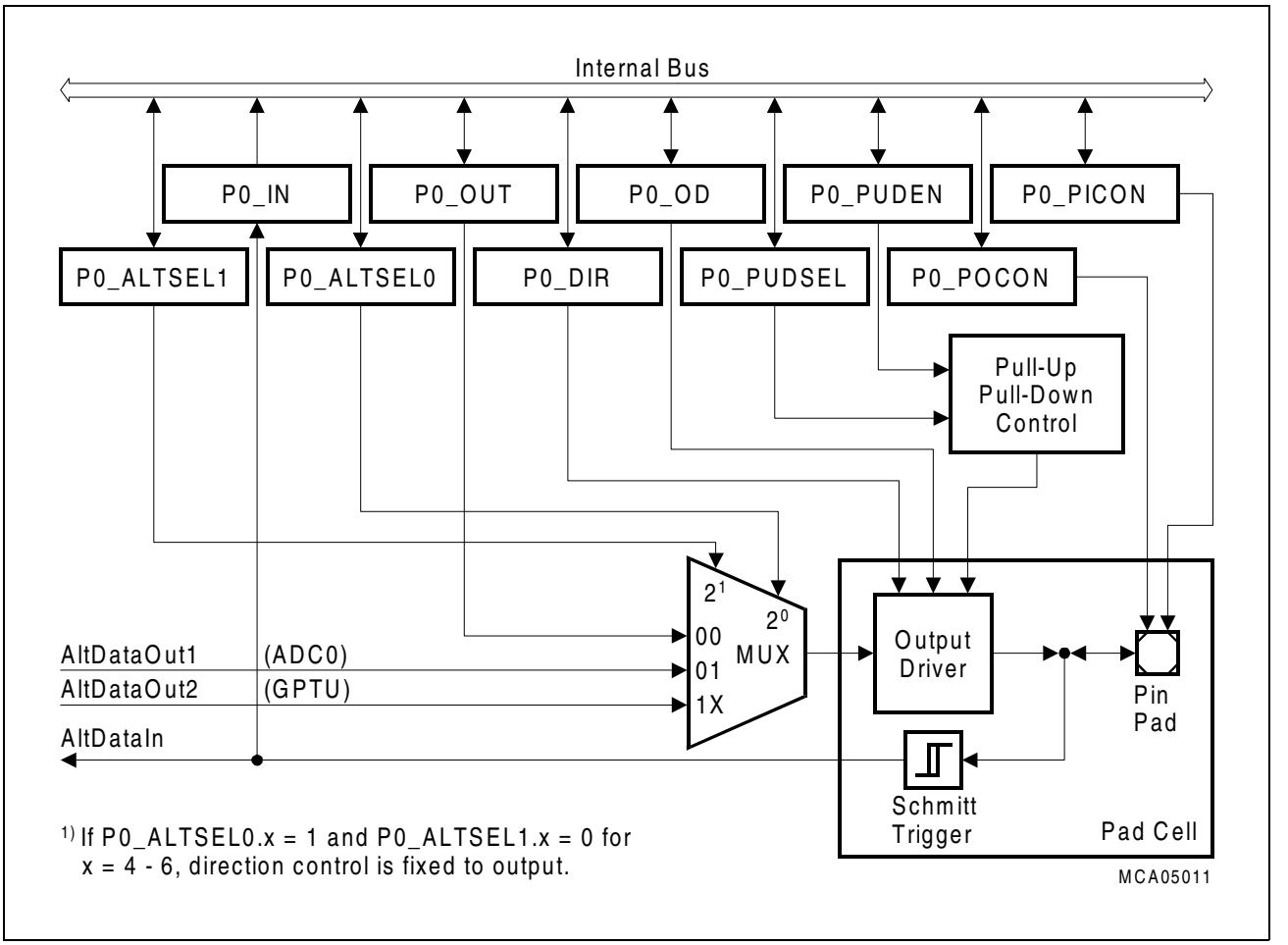


Figure 12-5 Port 0 Configuration P0[6:4]

Table 12-4 Port 0 Functions

Port Pin	Pin Functionality	Associated Register/Module	Alternate Function	Direction Control
P0.0	General purpose input	P0_IN.0	P0_ ALTSEL0.0 = 0 P0_ ALTSEL1.0 = X	P0_DIR.0 = 0
	General purpose output	P0_OUT.0		P0_DIR.0 = 1
	ADC0 external trigger input 0 AD0EXTIN0	ADC0		P0_DIR.0 = 0
	GPTU I/O line GPTU0 used as input	GPTU	P0_ ALTSEL0.0 = 1 P0_ ALTSEL1.0 = X	P0_DIR.0 = 0
	GPTU I/O line GPTU0 used as output			P0_DIR.0 = 1

Table 12-4 Port 0 Functions (cont'd)

Port Pin	Pin Functionality	Associated Register/Module	Alternate Function	Direction Control
P0.1	General purpose input	P0_IN.1	P0_ ALTSEL0.1 = 0 P0_ ALTSEL1.1 = X	P0_DIR.1 = 0
	ADC0 external trigger input 1 AD0EXTIN1	ADC0		
	DMA request input 0A DMAREQ0A	DMA		
	General purpose output	P0_OUT.1		P0_DIR.1 = 1
	GPTU I/O line GPTU1 used as input	GPTU	P0_ ALTSEL0.1 = 1 P0_ ALTSEL1.1 = X	P0_DIR.1 = 0
	GPTU I/O line GPTU1 used as output			P0_DIR.1 = 1
P0.2	General purpose input	P0_IN.2	P0_ ALTSEL0.2 = 0 P0_ ALTSEL1.2 = X	P0_DIR.2 = 0
	ADC1 external trigger input 0 AD1EXTIN0	ADC1		
	DMA request input 1A DMAREQ1A	DMA		
	General purpose output	P0_OUT.2		P0_DIR.2 = 1
	GPTU I/O line GPTU2 used as input	GPTU	P0_ ALTSEL0.2 = 1 P0_ ALTSEL1.2 = X	P0_DIR.2 = 0
	GPTU I/O line GPTU2 used as output			P0_DIR.2 = 1
P0.3	General purpose input	P0_IN.3	P0_ ALTSEL0.3 = 0 P0_ ALTSEL1.3 = X	P0_DIR.3 = 0
	General purpose output	P0_OUT.3		P0_DIR.3 = 1
	ADC1 external trigger input 1 AD1EXTIN1	ADC1		P0_DIR.3 = 0
	GPTU I/O line GPTU3 used as input	GPTU	P0_ ALTSEL03 = 1 P0_ ALTSEL1.3 = X	P0_DIR.3 = 0
	GPTU I/O line GPTU3 used as output			P0_DIR.3 = 1

Table 12-4 Port 0 Functions (cont'd)

Port Pin	Pin Functionality	Associated Register/Module	Alternate Function	Direction Control
P0.4	General purpose input	P0_IN.4	P0_ ALTSEL0.4 = 0 P0_ ALTSEL1.4 = 0	P0_DIR.4 = 0
	General purpose output	P0_OUT.4		P0_DIR.4 = 1
	ADC0 ext. multiplexer control line 0 AD0EMUX0	ADC	P0_ ALTSEL0.4 = 1 P0_ ALTSEL1.4 = 0	Output
	GPTU I/O line GPTU4 used as input	GPTU	P0_ ALTSEL0.4 = X P0_ ALTSEL1.4 = 1	P0_DIR.4 = 0
	GPTU I/O line GPTU4 used as output			P0_DIR.4 = 1
P0.5	General purpose input	P0_IN.5	P0_ ALTSEL0.5 = 0 P0_ ALTSEL1.5 = 0	P0_DIR.5 = 0
	General purpose output	P0_OUT.5		P0_DIR.5 = 1
	ADC0 ext. multiplexer control line 1 AD0EMUX1	ADC	P0_ ALTSEL0.5 = 1 P0_ ALTSEL1.5 = 0	Output
	GPTU I/O line GPTU5 used as input	GPTU	P0_ ALTSEL0.5 = X P0_ ALTSEL1.5 = 1	P0_DIR.5 = 0
	GPTU I/O line GPTU5 used as output			P0_DIR.5 = 1

Table 12-4 Port 0 Functions (cont'd)

Port Pin	Pin Functionality	Associated Register/Module	Alternate Function	Direction Control
P0.6	General purpose input	P0_IN.6	P0_ ALTSEL0.6 = 0 P0_ ALTSEL1.6 = 0	P0_DIR.6 = 0
	General purpose output	P0_OUT.6		P0_DIR.6 = 1
	ADC0 ext. multiplexer control line 2 AD0EMUX2	ADC	P0_ ALTSEL0.6 = 1 P0_ ALTSEL1.6 = 0	Output
	GPTU I/O line GPTU6 used as input	GPTU	P0_ ALTSEL0.6 = X P0_ ALTSEL1.6 = 1	P0_DIR.6 = 0
	GPTU I/O line GPTU6 used as output			P0_DIR.6 = 1
P0.7	General purpose input	P0_IN.7	P0_ ALTSEL0.7 = 0 P0_ ALTSEL1.7 = X	P0_DIR.7 = 0
	General purpose output	P0_OUT.7		P0_DIR.7 = 1
	ASC0 receiver input RXD0 used as input	ASC0	P0_ ALTSEL0.7 = 1 P0_ ALTSEL1.7 = X	P0_DIR.7 = 0
	ASC0 receiver input RXD0 used as output			P0_DIR.7 = 1
P0.8	General purpose input	P0_IN.8	P0_ ALTSEL0.8 = 0 P0_ ALTSEL1.7 = X	P0_DIR.8 = 0
	General purpose output	P0_OUT.8		P0_DIR.8 = 1
	ASC0 transmitter output TXD0	ASC0	P0_ ALTSEL0.8 = 1 P0_ ALTSEL1.7 = X	Output

Table 12-4 Port 0 Functions (cont'd)

Port Pin	Pin Functionality	Associated Register/Module	Alternate Function	Direction Control
P0.9	General purpose input	P0_IN.9	P0_ALTSEL0.9 = 0 P0_ALTSEL1.9 = X	P0_DIR.9 = 0
	General purpose output	P0_OUT.9		P0_DIR.9 = 1
	SSC0 clock input SCLK0	SSC0	P0_ALTSEL0.9 = 1 P0_ALTSEL1.9 = X	P0_DIR.9 = 0
	SSC0 clock output SCLK0			P0_DIR.9 = 1
P0.10	General purpose input	P0_IN.10	P0_ALTSEL0.10 = 0 P0_ALTSEL1.10 = X	P0_DIR.10 = 0
	General purpose output	P0_OUT.10		P0_DIR.10=1
	SSC0 master receive input MRST0	SSC0	P0_ALTSEL0.10 = 1 P0_ALTSEL1.10 = X	P0_DIR.10 = 0
	SSC0 slave transmit output MRST0			P0_DIR.10 = 1
P0.11	General purpose input	P0_IN.11	P0_ALTSEL0.11 = 0 P0_ALTSEL1.11 = X	P0_DIR.11 = 0
	General purpose output	P0_OUT.11		P0_DIR.11 = 1
	SSC0 slave receive input MTSR0	SSC0	P0_ALTSEL0.11 = 1 P0_ALTSEL1.11 = X	P0_DIR.11 = 0
	SSC0 master transmit output MTSR0			P0_DIR.11 = 1
P0.12	General purpose input	P0_IN.12	P0_ALTSEL0.12 = 0 P0_ALTSEL1.12 = X	P0_DIR.12 = 0
	General purpose output	P0_OUT.12		P0_DIR.12 = 1
	CAN receiver input 0 RXDCAN0	CAN	P0_ALTSEL0.12 = 1 P0_ALTSEL1.12 = X	Input

Table 12-4 Port 0 Functions (cont'd)

Port Pin	Pin Functionality	Associated Register/Module	Alternate Function	Direction Control
P0.13	General purpose input	P0_IN.13	P0_ ALTSEL0.13 = 0 P0_ ALTSEL1.13 = X	P0_ DIR.13 = 0
	General purpose output	P0_OUT.13		P0_ DIR.13 = 1
	CAN transmitter output 0 TXDCAN0	CAN	P0_ ALTSEL0.13 = 1 P0_ ALTSEL1.13 = X	Output
P0.14	General purpose input	P0_IN.14	P0_ ALTSEL0.14 = 0 P0_ ALTSEL1.14 = X	P0_ DIR.14 = 0
	General purpose output	P0_OUT.14		P0_ DIR.14 = 1
	CAN receiver input 1 RXDCAN1	CAN	P0_ ALTSEL0.14 = 1 P0_ ALTSEL1.14 = X	Input
P0.15	General purpose input	P0_IN.15	P0_ ALTSEL0.15 = 0 P0_ ALTSEL1.15 = X	P0_ DIR.15 = 0
	General purpose output	P0_OUT.15		P0_ DIR.15 = 1
	CAN transmitter output 1 TXDCAN1	CAN	P0_ ALTSEL0.15 = 1 P0_ ALTSEL1.15 = X	Output

12.4 Port 1

Port 1 is a 16-bit bidirectional I/O port. It serves as GPIO lines or as input or output lines 0 - 15 of the General Purpose Timer Array (GPTA). A port line of Port 1 can be used as standard GPIO pin if it is not used for the GPTA.

12.4.1 Features

- Push/pull output drivers
- Optional Open Drain Output Mode (programmable via register P1_OD)
- 5.0 Volt operation for GPIO
- Pin-wise programmable input threshold (TTL or CMOS-like) via register P1_PICON
- Programmable slew-rate and output driver strength via register P1_POCON
- Programmable pull-up/pull-down devices via registers P1_PUDSEL and P1_PUDEN

12.4.2 Registers

The following port registers are available at Port 1:

Table 12-5 Port 1 Registers

Register Short Name	Register Long Name
P1_OUT	Port 1 Data Output Register
P1_IN	Port 1 Data Input Register
P1_DIR	Port 1 Direction Register
P1_OD	Port 1 Open Drain Mode Register
P1_PUDSEL	Port 1 Pull-up/Pull-down Select Register
P1_PUDEN	Port 1 Pull-up/Pull-down Enable Register
P1_POCON	Port 1 Port Output Characteristics Control Register
P1_PICON	Port 1 Input Configuration Register

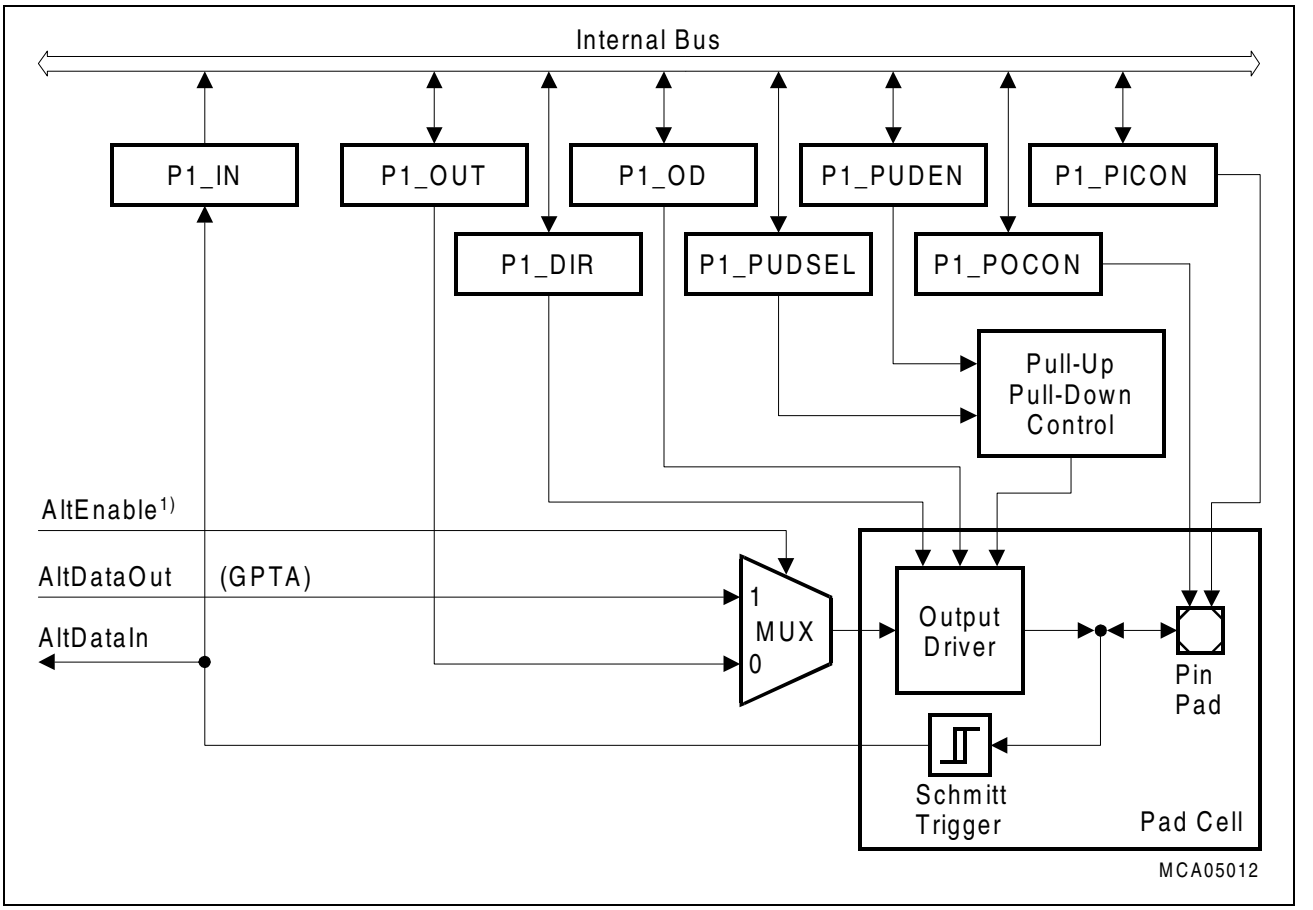


Figure 12-6 Port 1 Configuration

Table 12-6 Port 1 Functions

Port Pin	Pin Functionality	Associated Register/Modul	Alternate Function	Direction Control
P1.x (x = 15-0)	General purpose input	P1_IN.x	AltEnable = 0 1)	P1_DIR.x = 0
	General purpose output	P1_OUT.x		P1_DIR.x = 1
	GPTA input INx	GPTA	AltEnable = 1 1)	P1_DIR.x = 0
	GPTA output OUTx			P1_DIR.x = 1

1) Alternate functions of Port 1 lines used for GPTA I/O alternate function are controlled directly by the GPTA module via the AltEnable signal.

12.5 Port 2

Port 2 is a 16-bit bidirectional I/O port. It serves as GPIO lines or as input or output lines 31 - 16 of the General Purpose Timer Array (GPTA). A port line of Port 2 can be used as standard GPIO pin if it is not used for the GPTA.

12.5.1 Features

- Push/pull output drivers
- Optional Open Drain Output Mode (programmable via register P2_OD)
- 5.0 Volt operation for GPIO
- Pin-wise programmable input threshold (TTL or CMOS-like) via register P2_PICON
- Programmable slew-rate and output driver strength via register P2_POCON
- Programmable pull-up/pull-down devices via registers P2_PUDSEL and P2_PUDEN

12.5.2 Registers

The following port registers are available at Port 2:

Table 12-7 Port 2 Registers

Register Short Name	Register Long Name
P2_OUT	Port 2 Data Output Register
P2_IN	Port 2 Data Input Register
P2_DIR	Port 2 Direction Register
P2_OD	Port 2 Open Drain Mode Register
P2_PUDSEL	Port 2 Pull-up/Pull-down Select Register
P2_PUDEN	Port 2 Pull-up/Pull-down Enable Register
P2_POCON	Port 2 Port Output Characteristics Control Register
P2_PICON	Port 2 Input Configuration Register

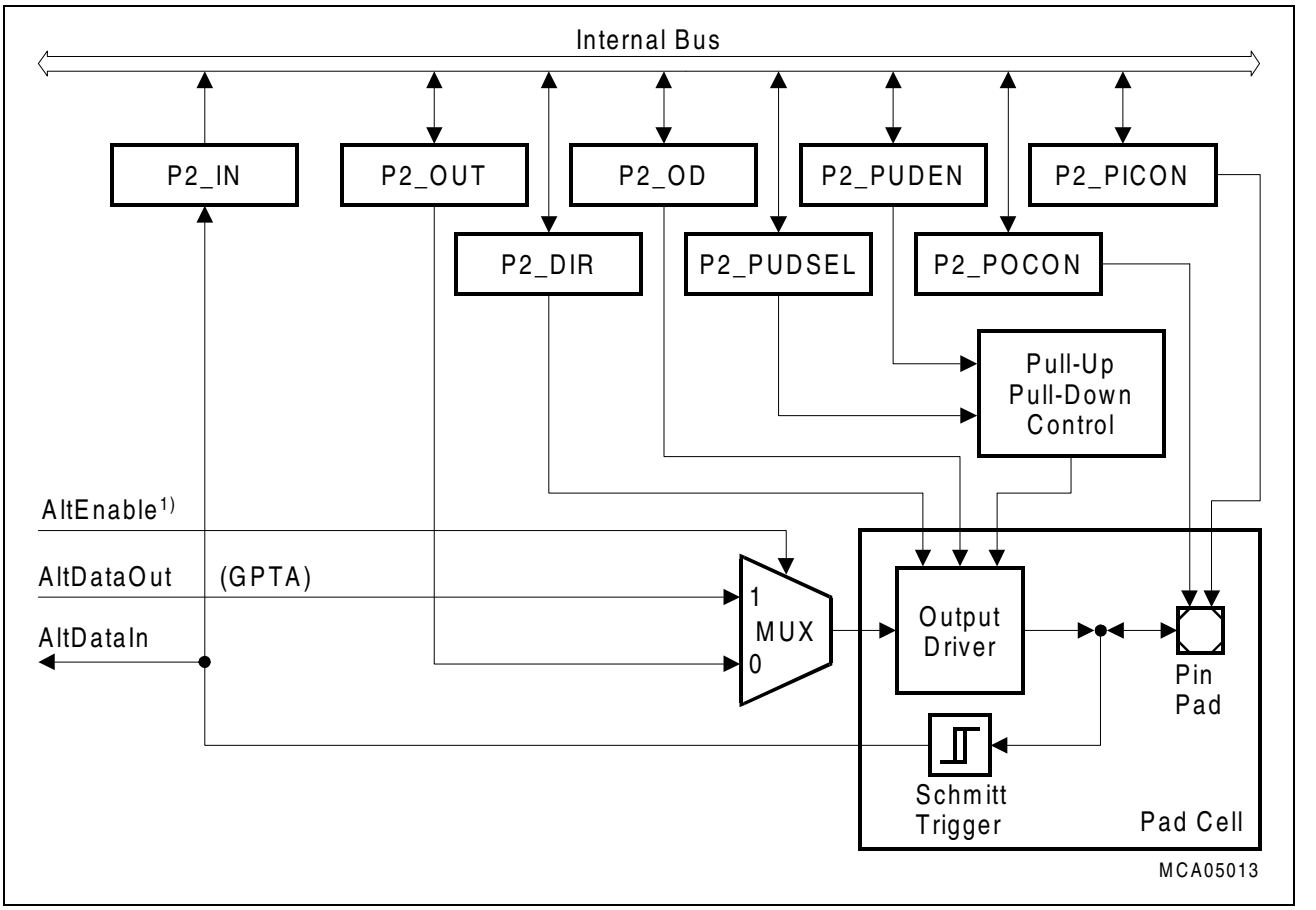


Figure 12-7 Port 2 Configuration

Table 12-8 Port 2 Functions

Port Pin	Pin Functionality	Associated Register/Modul	Alternate Function	Direction Control
P2.x (x = 15-0) (y = x + 16)	General purpose input	P2_IN.x	AltEnable = 0 1)	P2_DIR.x = 0
	General purpose output	P2_OUT.x		P2_DIR.x = 1
	GPTA input INy	GPTA	AltEnable = 1 1)	P2_DIR.x = 0
	GPTA output OUTy			P2_DIR.x = 1

1) Alternate functions of Port 2 lines used for GPTA I/O alternate function are controlled directly by the GPTA module via the AltEnable signal.

12.6 Port 3

Port 3 is a 16-bit bidirectional I/O port. It serves as GPIO lines or as input or output lines 47 - 32 of the General Purpose Timer Array (GPTA). A port line of Port 3 can be used as standard GPIO pin if it is not used for the GPTA.

12.6.1 Features

- Push/pull output drivers
- Optional Open Drain Output Mode (programmable via register P3_OD)
- 5.0 Volt operation for GPIO
- Pin-wise programmable input threshold (TTL or CMOS-like) via register P3_PICON
- Programmable slew-rate and output driver strength via register P3_POCON
- Programmable pull-up/pull-down devices via registers P3_PUDSEL and P3_PUDEN

12.6.2 Registers

The following port registers are available at Port 3:

Table 12-9 Port 3 Registers

Register Short Name	Register Long Name
P3_OUT	Port 3 Data Output Register
P3_IN	Port 3 Data Input Register
P3_DIR	Port 3 Direction Register
P3_OD	Port 3 Open Drain Mode Register
P3_PUDSEL	Port 3 Pull-up/Pull-down Select Register
P3_PUDEN	Port 3 Pull-up/Pull-down Enable Register
P3_POCON	Port 3 Port Output Characteristics Control Register
P3_PICON	Port 3 Input Configuration Register

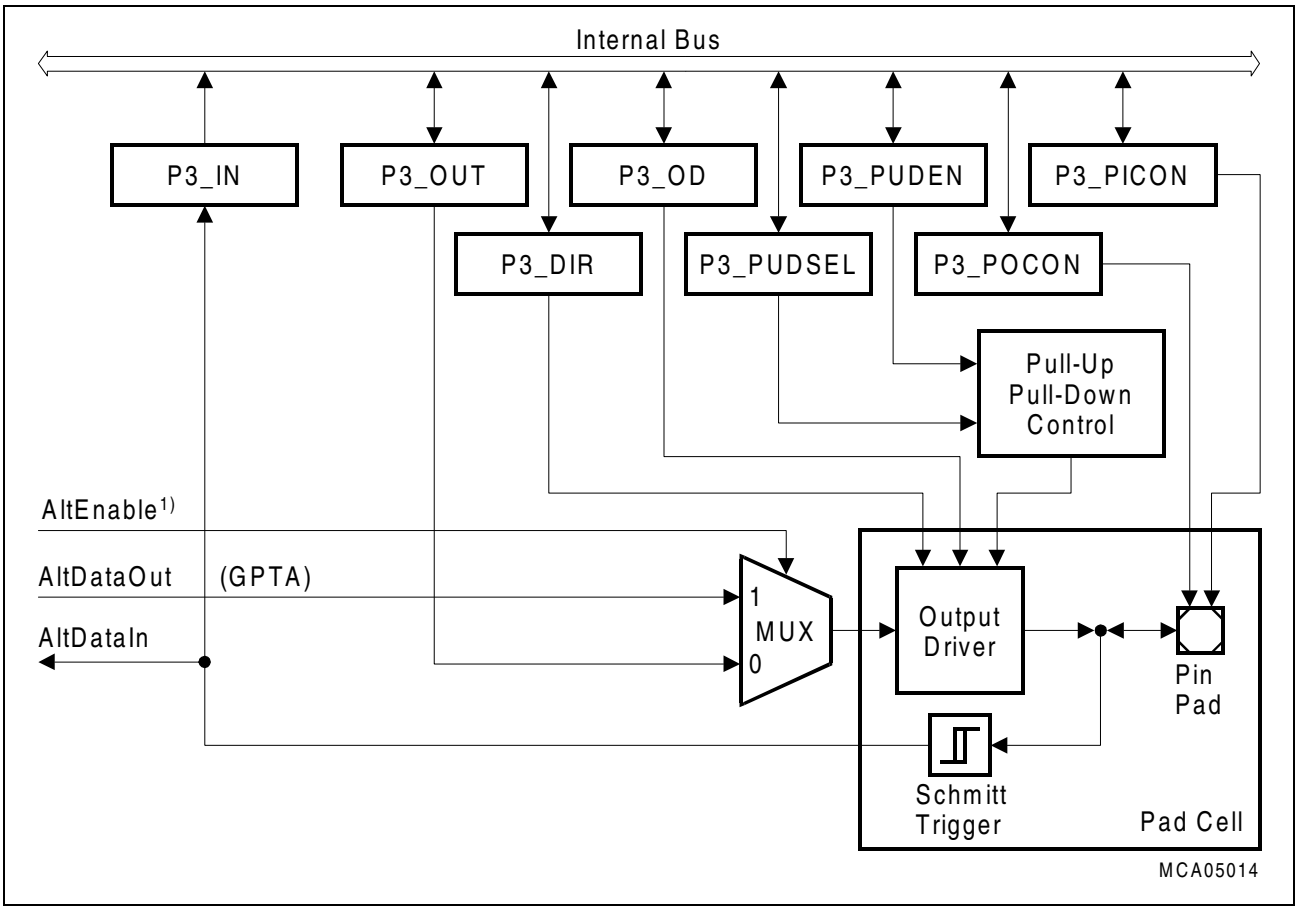


Figure 12-8 Port 3 Configuration

Table 12-10 Port 3 Functions

Port Pin	Pin Functionality	Associated Register/Modul	Alternate Function	Direction Control
P3.x (x = 15-0) (y = x + 32)	General purpose input	P3_IN.x	AltEnable = 0 1)	P3_DIR.x = 0
	General purpose output	P3_OUT.x		P3_DIR.x = 1
	GPTA input INy	GPTA	AltEnable = 1 1)	P3_DIR.x = 0
	GPTA output OUTy			P3_DIR.x = 1

1) Alternate functions of Port 3 lines used for GPTA I/O alternate function are controlled directly by the GPTA module via the AltEnable signal.

12.7 Port 4

Port 4 is an 8-bit bidirectional I/O port. It serves as GPIO lines or as input or output lines 55 - 48 of the General Purpose Timer Array (GPTA). During hardware reset the Port 4 lines are also used as start-up configuration selection inputs. A port line of Port 4 can be used as standard GPIO pin if it is not used for the GPTA.

12.7.1 Features

- Push/pull output drivers
- Optional Open Drain Output Mode (programmable via register P4_OD)
- 5.0 Volt operation for GPIO
- Pin-wise programmable input threshold (TTL or CMOS-like) via register P4_PICON
- Programmable slew-rate and output driver strength via register P4_POCON
- Programmable pull-up/pull-down devices via registers P4_PUDSEL and P4_PUDEN

12.7.2 Registers

The following port registers are available at Port 4:

Table 12-11 Port 4 Registers

Register Short Name	Register Long Name
P4_OUT	Port 4 Data Output Register
P4_IN	Port 4 Data Input Register
P4_DIR	Port 4 Direction Register
P4_OD	Port 4 Open Drain Mode Register
P4_PUDSEL	Port 4 Pull-up/Pull-down Select Register
P4_PUDEN	Port 4 Pull-up/Pull-down Enable Register
P4_POCON	Port 4 Port Output Characteristics Control Register
P4_PICON	Port 4 Input Configuration Register

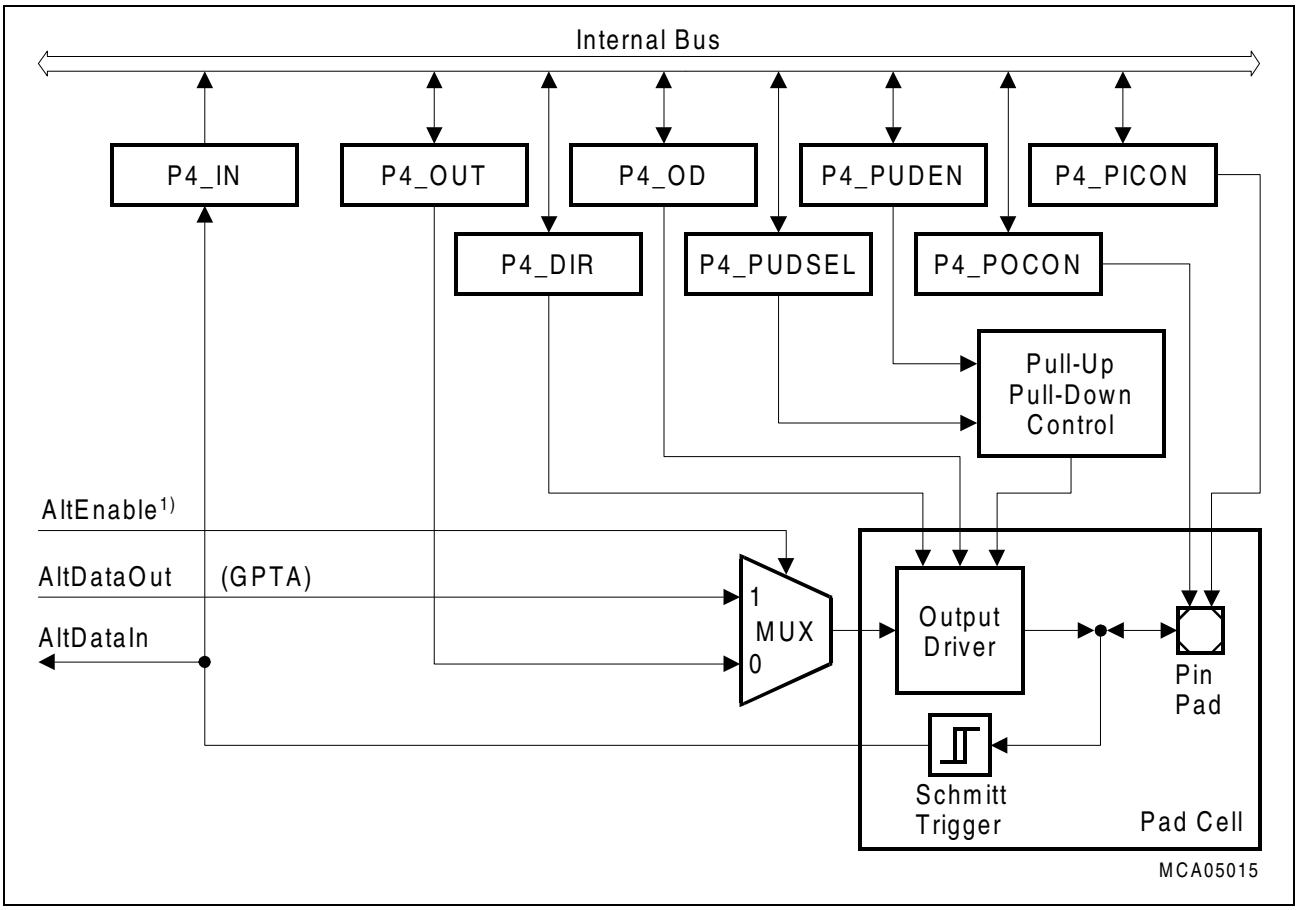


Figure 12-9 Port 4 Configuration

Table 12-12 Port 4 Functions

Port Pin	Pin Functionality	Associated Register/Modul	Alternate Function	Direction Control
P4.x (x = 7-0) (y = x + 48)	General purpose input	P4_IN.x	AltEnable = 0 1)	P4_DIR.x = 0
	DMA request input2)	DMA		
	General purpose output	P4_OUT.x		P4_DIR.x = 1
	GPTA input INy	GPTA	AltEnable = 1 1)	P4_DIR.x = 0
	GPTA output OUTy			P4_DIR.x = 1

1) Alternate functions of Port 4 lines used for GPTA I/O alternate function are controlled directly by the GPTA module via the AltEnable signal.

2) For x = 1, the DMA request input is DMAREQ0B; for x = 2, the DMA request input is DMAREQ1B;

12.8 Port 5

Port 5 is a 5-bit bidirectional I/O port. It serves as GPIO lines or as alternate function I/O lines for the Asynchronous/Synchronous Serial Interface ASC1 or the High-Speed Synchronous Serial Interface SSC1. A port line can be used as standard GPIO pin if it is not used for ASC1 or SSC1.

12.8.1 Features

- Push/pull output drivers
- Optional Open Drain Output Mode (programmable via register P5_OD)
- 5.0 Volt operation for GPIO
- Pin-wise programmable input threshold (TTL or CMOS-like) via register P5_PICON
- Programmable slew-rate and output driver strength via register P5_POCON
- Programmable pull-up/pull-down devices via registers P5_PUDSEL and P5_PUDEN

12.8.2 Registers

The following port registers are available at Port 5:

Table 12-13 Port 5 Registers

Register Short Name	Register Long Name
P5_OUT	Port 5 Data Output Register
P5_IN	Port 5 Data Input Register
P5_DIR	Port 5 Direction Register
P5_OD	Port 5 Open Drain Mode Register
P5_PUDSEL	Port 5 Pull-up/Pull-down Select Register
P5_PUDEN	Port 5 Pull-up/Pull-down Enable Register
P5_POCON	Port 5 Port Output Characteristics Control Register
P5_PICON	Port 5 Input Configuration Register
P5_ALTSEL0	Port 5 Alternate Select Register 0

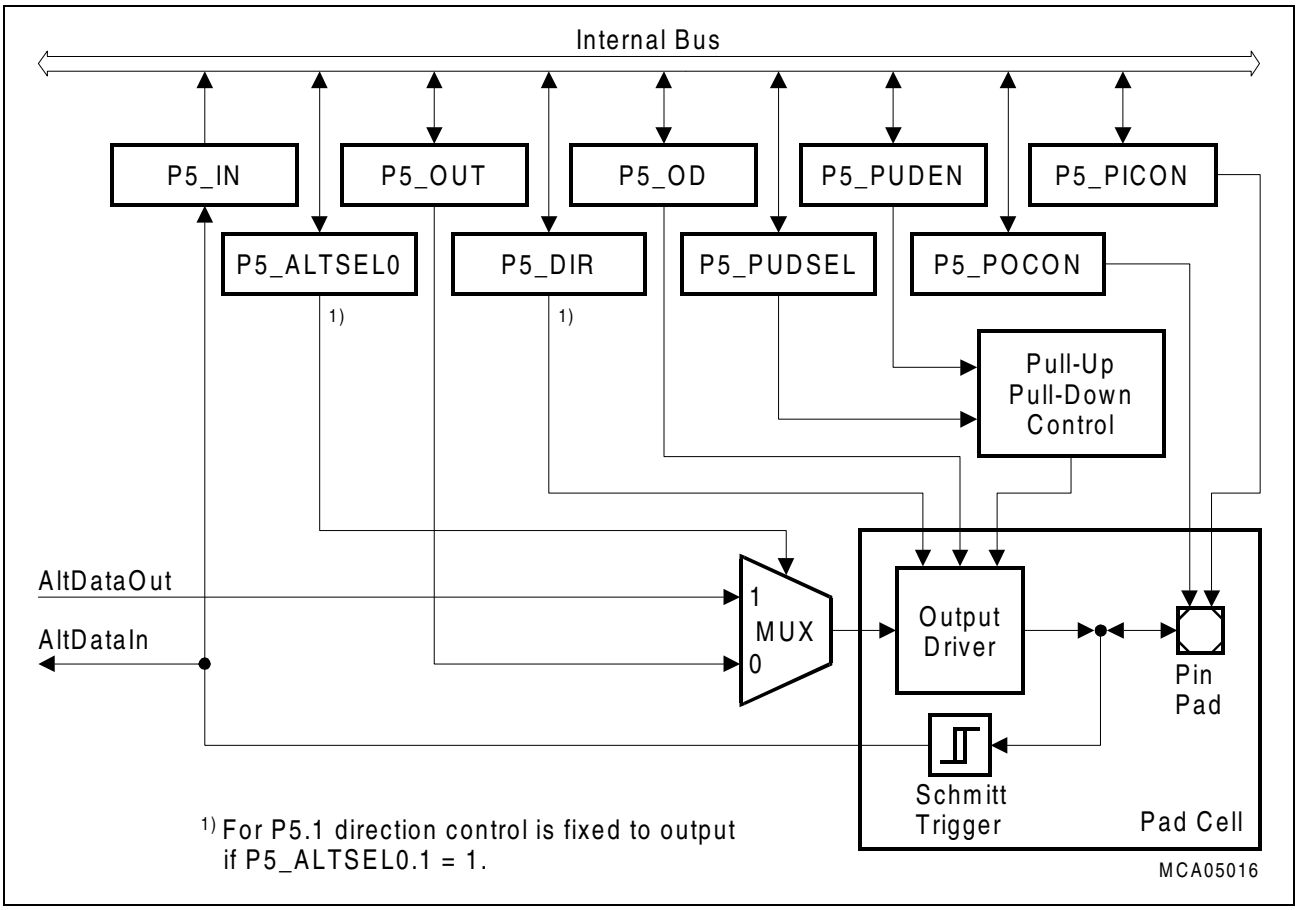


Figure 12-10 Port 5 Configuration

Table 12-14 Port 5 Functions

Port Pin	Pin Functionality	Associated Register/Modul	Alternate Function	Direction Control
P5.0	General purpose input	P5_IN.0	P5_ ALTSEL0.0 = 0	P5_DIR.0 = 0
	DMA request input 0C DMAREQ0C	DMA		
	General purpose output	P5_OUT.0		P5_DIR.0 = 1
	ASC1 receiver input RXD1 used as input	ASC1	P5_ ALTSEL0.0 = 1	P5_DIR.0 = 0
	ASC1 receiver input RXD1 used as output			P5_DIR.0 = 1

Table 12-14 Port 5 Functions

Port Pin	Pin Functionality	Associated Register/Modul	Alternate Function	Direction Control
P5.1	General purpose input	P5_IN.12	P5_ ALTSEL0.1 = 0	P5_DIR.1 = 0
	DMA request input 1C DMAREQ1C	DMA		
	General purpose output	P5_OUT.12		P5_DIR.1 = 1
	ASC1 transmitter output TXD1	ASC1	P5_ ALTSEL0.1 = 1	Output
P5.2	General purpose input	P5_IN.2	P5_ ALTSEL0.2 = 0	P5_DIR.2 = 0
	General purpose output	P5_OUT.2		P5_DIR.2 = 1
	SSC1 clock input SCLK1	SSC1	P5_ ALTSEL0.2 = 1	P5_DIR.2 = 0
	SSC1 clock output SCLK1			P5_DIR.2 = 1
P5.3	General purpose input	P5_IN.3	P5_ ALTSEL0.3 = 0	P5_DIR.3 = 0
	General purpose output	P5_OUT.3		P5_DIR.3 = 1
	SSC1 master receive input MRST1	SSC1	P5_ ALTSEL0.3 = 1	P5_DIR.3 = 0
	SSC1 slave transmit output MRST1			P5_DIR.3 = 1
P5.4	General purpose input	P5_IN.4	P5_ ALTSEL0.4 = 0	P5_DIR.4 = 0
	General purpose output	P5_OUT.4		P5_DIR.4 = 1
	SSC1 slave receive input MTSR1	SSC1	P5_ ALTSEL0.4 = 1	P5_DIR.4 = 0
	SSC1 master transmit output MTSR1			P5_DIR.4 = 1

12.9 Trace Port (TC1765T only)

The Trace Port TP[15:0] is the OCDS Level 2 trace output port and is only available in the TC1765T. In general, the Trace Port has following features:

- Output only
- High current mode driver characteristic (see Px_POCON.PDC on **Page 12-13**)
- Normal timing edge characteristic (see Px_POCON.PEC on **Page 12-13**)
- Synchronous to clock CPUCLK (see also **Chapter 19**)
- No pull-up resistors

12.10 External Bus Interface

The EBU has several dedicated I/O lines which are grouped into:

- Address bus A[23:0] (outputs)
- Data bus D[31:0] (inputs/outputs)
- Control lines (inputs/outputs

The EBU I/O lines have the input/output characteristics as shown in **Table 12-15**.

Table 12-15 EBU I/O Line Characteristics

EBU I/O Line		Driver Characteristic	Edge Characteristic	Input Threshold
Description	**In/Out**			
A[23:0]	O	High Current Mode as described for Px_POCON.PDCn = 0_B	Normal timing as described for Px_POCON.PECn = 0_B	–
D[31:0]	I/O			CMOS-like
$\overline{CS}$[3:0] $\overline{CSEMU}$/$\overline{CSOVL}$ $\overline{BC}$[3:0] $\overline{RD}$ $\overline{RD}$/$\overline{WR}$ $\overline{ADV}$ $\overline{BAA}$ $\overline{CODE}$	O			–
$\overline{WAIT}$/$\overline{IND}$	I	–	–	CMOS-like

All EBU I/O lines are equipped with internal pull-up devices. These internal pull-up/pull-down devices can be enabled group-wise via control bits which are located in register SCU_CON:

- SCU_CON.EBUAPEN: controls pull-up devices at EBU address lines A[23:0]
- SCU_CON.EBUDPEN: controls pull-up devices at EBU data bus lines D[31:0]
- SCU_CON.EBUCPEN: controls pull-up devices at EBU control lines

For more details see **Chapter 4** of this TC1765 System Units User's Manual.

12.11 ADC Analog Input Port

The ADC Analog Input Port provides 24 analog input lines for the A/D converters ADC0 and ADC1. More details on the analog input lines are described in chapter “Analog Digital Converters (ADC0, ADC1)” of the TC1765 Peripheral Units User’s Manual.

12.12 Electrical Characteristics

Pin Classes

The TC1765 has two classes of digital I/O pins:

- **Class A** pins, which are 3.0 - 5.5 V voltage pins
- **Class B** pins, which are 2.5 V nominal voltage pins (input tolerant for 3.3 V)
- **Class C** pins, which are 2.5 V nominal voltage pins only

Table 12-16 shows the assignments of all digital I/O pins to pin classes and to V_{DD} power supply pins.

Table 12-16 Assignments of Digital Pins to Pin Classes and Power Supply Pins

Pins	Pin Classes	Power Supply	
Port 0 to Port 5, BYPASS, $\overline{\text{HDRST}}$	Class A (3.0 - 5.5 V)	V_{DDP}	V_{SS}
D[31:0], A[23:0], $\overline{\text{CS}}$[3:0], $\overline{\text{CSEMU}}$/$\overline{\text{CSOVL}}$, $\overline{\text{BC}}$[3:0], $\overline{\text{RD}}$, RD/$\overline{\text{WR}}$, ADV, $\overline{\text{WAIT}}$/$\overline{\text{IND}}$, $\overline{\text{BAA}}$, $\overline{\text{CODE}}$, $\overline{\text{TRST}}$, TCK, TDI, TDO, TMS, $\overline{\text{ODCSE}}$, $\overline{\text{BRKIN}}$, $\overline{\text{BRKOUT}}$, $\overline{\text{NMI}}$, $\overline{\text{PORST}}$, ECOUT, ECIN, CPUCLK, $\overline{\text{TESTMODE}}$, TP[15:0]	Class B (nominal 2.5 V)	V_{DD}	
XTAL1, XTAL2	Class C (nominal 2.5 V)	V_{DDOSC}	V_{SSOSC}
no pins assigned	(nominal 2.5 V)	V_{DDSRAM} $V_{DDSBRAM}$	V_{SS}

Note: Detailed electrical AC and DC characteristics are defined in the TC1765 Data Sheet.

13 External Bus Unit

The External Bus Unit (EBU) of the TC1765 is the interface between external memories and peripheral units and the internal memories and peripheral units. The basic structure of the EBU is shown in **Figure 13-1**.

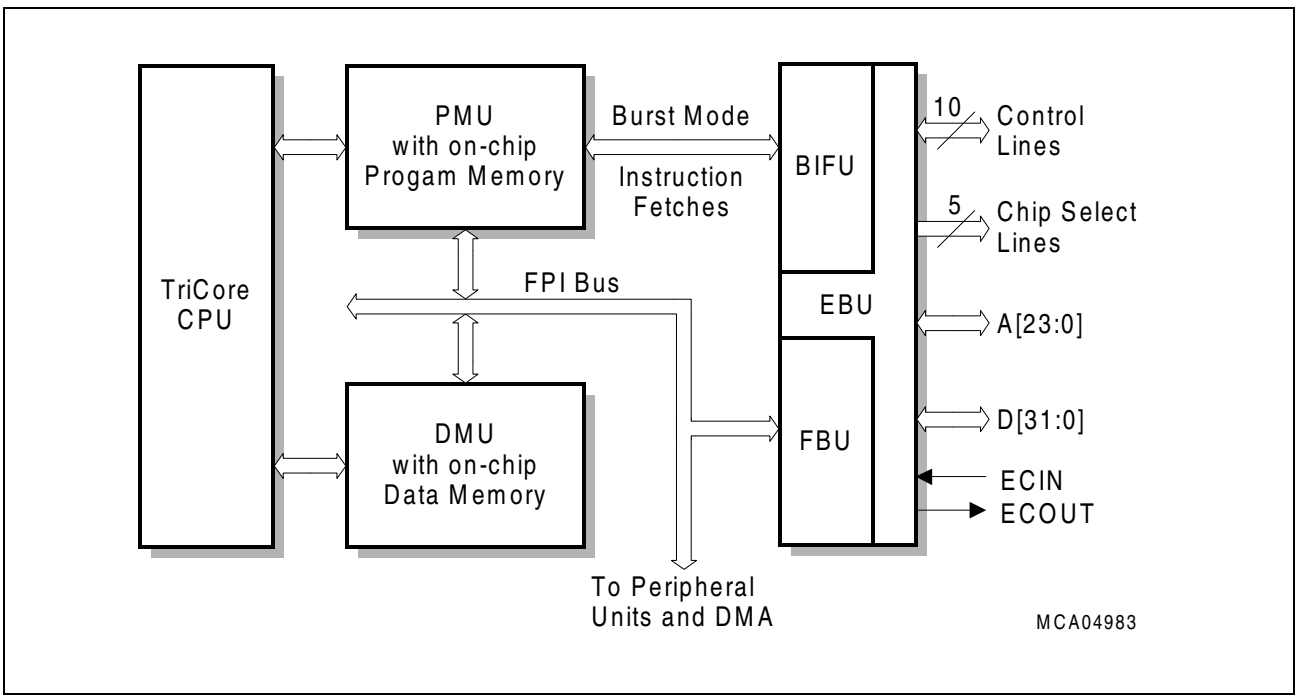


Figure 13-1 EBU Structure and Interfaces

The EBU consists of two parts and is used for the following two operations:

- FBU (FPI Bus Unit):
  - Communication with external memories or peripheral units via the FPI Bus
  - Non-burst instruction fetches
- BIFU (Burst Instruction Fetch Unit):
  - Instruction fetches from the PMU to external Burst Flash program memories with 16-bit and 32-bit data width

The EBU controls all transactions required for these two operations and in particular handles the arbitration between these two tasks.

The types of external devices/Bus modes controlled by the FBU are:

- INTEL style peripherals (separate $\overline{RD}$ and $\overline{WR}$ signals)
- MOTOROLA style peripherals ($\overline{OE}$ and R/$\overline{W}$)
- ROMs, EPROMs
- Static RAMs
- Peripherals with demultiplexed A/D bus
- Burst Mode Flash Memories
- 8-, 16- and 32-bit data bus width

13.1 Overview

The external bus established by the EBU consists of a 32-bit wide data bus, a 24-bit wide address bus, and a number of control signals. With four user chip select lines, four external address ranges can be accessed, each with a size of up to 16 MBytes (besides the special emulator range). Each of these ranges can be programmed individually in terms of location, size and access parameters (such as data size, address mode, wait states, etc.), making it possible to connect and access different device types in one system. The EBU dynamically adjusts the access sequence according to the programmed parameters for each selectable device.

The EBU also provides special support for external emulator hardware, debugging and instrumentation support.

13.2 EBU Feature Summary

- 32-bit wide data bus (D[31:0])
  - Data width of external device can be 8, 16 or 32 bits
  - Automatic data assembly/disassembly operation
  - Non-multiplexed operation
- 24-bit wide address bus (A[23:0])
- Bus control signals
  - EBU clock output (ECOUT)
  - EBU clock input (ECIN)
  - Read ($\overline{RD}$) and read/write (RD/$\overline{WR}$)
  - Four byte control signals ($\overline{BC}$[3:0])
  - Four user chip selects ($\overline{CS}$[3:0])
  - External synchronous/asynchronous wait state control ($\overline{WAIT}$/$\overline{IND}$)
  - Code/data fetch indication ($\overline{CODE}$)
  - Address valid ($\overline{ADV}$)
  - Burst Address Advance ($\overline{BAA}$)
- 4 user address ranges
  - Programmable location and size
  - Individual chip select for each range
  - Programmable mirror function: the same physical device can be accessed in two different address ranges
  - Enable/disable control
- Programmable access parameters for each address range
  - Data width
  - Byte control signal operation
  - Address setup and hold timing
  - Data hold wait states
  - Read/write wait states
  - Recovery cycle wait states
  - External $\overline{WAIT}$/$\overline{IND}$ input enable and active level control, asynchronous or synchronous operation
  - Write protection for region
- Programmable wait state insertion to meet recovery/tri-state time needs of external devices between
  - Read and write accesses
  - Accesses to different address ranges
- Automatic self-configuration on boot from external memory
  - Reads configuration data from external memory
- Dedicated emulation support
  - Emulator address range
  - Emulator/Debug/Instrumentation memory chip select ($\overline{CSEMU}$/$\overline{CSOVL}$)
  - Special boot from emulation memory

13.3 Basic EBU Operation

The EBU is the interface or gateway from the internal on-chip system onto the external on-board system. **Figure 13-2** shows an example for the connection of an external system.

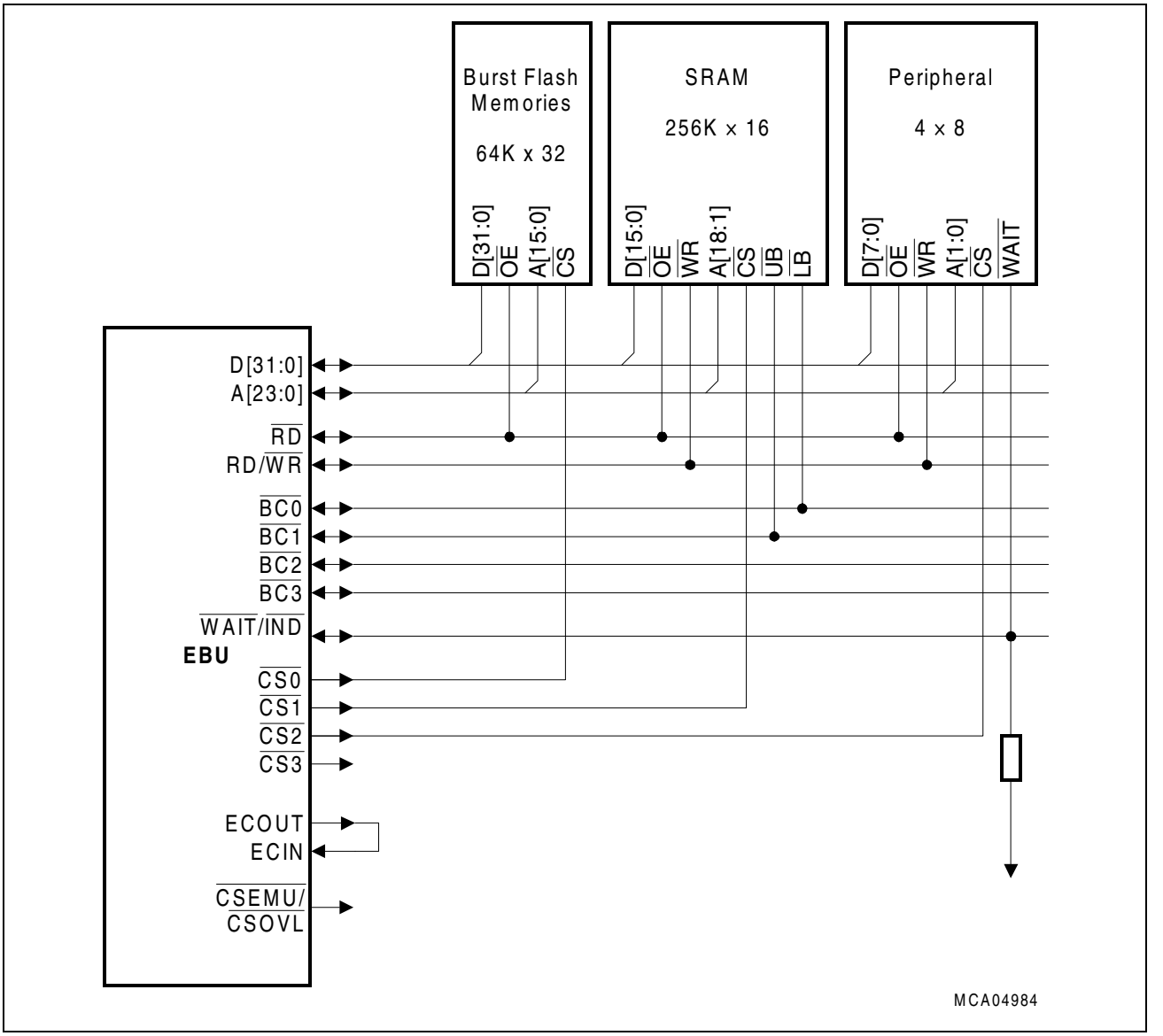


Figure 13-2 Example Configuration for Connection of External Devices

*Note: The example given in **Figure 13-2** is valid for small systems with a low capacity (approximately 50 pF max.). For larger systems and high frequency applications, the external bus must be separated by additional buffers into a fast section (that is, low capacity, maximum capacitance of 50 pF, and 0 wait-states) and a slow section (that is, high capacity, with wait states).*

The basic operation of the EBU in these two fundamental modes is described in the following sections.

13.4 EBU Signal Description

The external signals of the EBU are listed in **Table 13-1** and described in the following section.

Table 13-1 EBU Signals Available on the TC1765 Ports

Signal	Type	Function	Used by	
			FBU	BIFU
D[31:0]	O	Data bus lines 31-0	X	X
A[23:0]	O	Address bus lines 23-0	X	X
$\overline{CS0}$	O	Chip select 0	X	X
$\overline{CS[3:1]}$	O	Chip select line n (n = 3-1)	X	–
$\overline{CSEMU}$/ $\overline{CSOVL}$	O	Chip select for emulation region selects external emulator memory region	X	–
		Chip select for overlay memory selects external overlay memory region	X	–
ECOUT	O	EBU clock output	X	X
ECIN	I	EBU clock input	X	X
$\overline{RD}$	O	Read control line; active during read operation	X	X
RD/$\overline{WR}$	O	Write control line; active during write operation	X	–
$\overline{ADV}$	O	Address valid strobe	X	X
$\overline{BC0}$	O	Byte control line n (n = 3-0) controls the byte access to corresponding byte location	X	–
$\overline{BC1}$	O		X	–
$\overline{BC2}$	O		X	–
$\overline{BC3}$	O		X	–
$\overline{WAIT}$/$\overline{IND}$	I	Wait input/End of burst input	X	X
$\overline{BAA}$	O	Burst address advance output	–	X
$\overline{CODE}$	O	Code fetch status output	X	X

*Note: Pull-up resistors can be switched groupwise to each pin of the three EBU I/O line groups: A[23:0], D[31:0], EBU control lines. After reset the EBU control lines have pull-up resistors enabled (see also description of SCU_CON register in **Chapter 4**).*

13.4.1 EBU Clock Output, ECOUT

The EBU clock of the TC1765 is provided at pin ECOUT for timing purposes (timing reference). ECOUT can be enabled/disabled by clearing/setting bit ECOUTDIS in register SCU_CON (see **Chapter 4**).

13.4.2 EBU Clock Input, ECIN

The EBU clock input of the TC1765 is used to latch read data into the EBU. Normally it should be directly connected to ECOUT to compensate the ECOUT clock pad delay.

13.4.3 Address Bus, A[23:0]

The address bus of the EBU consists of 24 address lines, giving a directly addressable range of 16 MBytes. Directly addressable means that these address lines can be used to access any location within one external device, such as a memory. This external device is selected via one of the chip select lines. While there are four chip selects, four such devices with up to 64 MBytes of address range can be used in the external system.

13.4.4 Data Bus, D[31:0]

The data bus transfers data information. The width of this bus is 32 bits. External devices with 8, 16 or 32 bits of data width can be connected to the data bus. Burst mode instruction fetches can be performed with 16-bit or 32-bit data bus width.

The EBU adjusts the data on the data bus to the width of the external device, according to the programmed parameters in its control registers. See **Section 13.7.3** for more information. The byte control signals, $\overline{BC}x$, specify which part of the data bus carries valid data. See also **Section 13.4.6**.

13.4.5 Read/Write Strobes, $\overline{RD}$ and RD/$\overline{WR}$

Two lines are provided to trigger the read ($\overline{RD}$) and write (RD/$\overline{WR}$) operations of external devices. While some read/write devices require both signals, there are devices with only one control input. The RD/$\overline{WR}$ line is then used for these devices. This line will go to an active low level on a write, and will stay inactive high on a read. The external device should only evaluate this signal in conjunction with an active chip select. Thus, an active chip select in combination with a high level on the RD/$\overline{WR}$ line indicates a read access to this device.

13.4.6 Byte Control Signals, $\overline{BCx}$

The byte control signals $\overline{BC}[3:0]$ select the appropriate byte lanes of the data bus for both read and write accesses. **Table 13-2** shows the activation on access to a 32-bit, 16-bit or 8-bit external device. Please note that this scheme supports little-endian devices.

Table 13-2 Byte Control Pin Usage

Width of External Device	BC3	BC2	BC1	BC0
32-bit device with byte write capability	D[31:24]	D[23:16]	D[15:8]	D[7:0]
16-bit device with byte write capability	inactive (high)	inactive (high)	D[15:8]	D[7:0]
8-bit device	inactive (high)	inactive (high)	inactive (high)	D[7:0]

Note: The $\overline{BCx}$ lines are not activated during burst mode cycles executed by the BIFU.

Signals $\overline{BCx}$ can be programmed for different timing. The available modes cover a wide range of external devices, such as RAM with separate byte write-enable signals, and RAM with separate byte chip-select signals. This allows external devices to connect without any external “glue” logic. Refer to **Table 13-3** for byte-control timing.

Table 13-3 Byte Control Signal Timing Options

Programmed Mode	$\overline{BCx}$ Signal Timing
Chip Select Mode	$\overline{BCx}$ signals have the same timing as the generated chip select $\overline{CS}$.
Control Mode	$\overline{BCx}$ signals have the same timing as the generated control signals $\overline{RD}$ or RD/$\overline{WR}$.
Write Enable Mode	$\overline{BCx}$ signals have the same timing as the generated control signal RD/$\overline{WR}$.

13.4.7 Variable Wait State Control, Burst Abort $\overline{WAIT}$/$\overline{IND}$

This is an input signal to the EBU serving to purposes. Normally it is used to dynamically insert wait states into read or write data cycles controlled by the memory or the peripheral component.

The BIFU can also use this input as a burst abort control signal input.

A detailed description of the different functionalities is given in **Chapter 13.6** and **Chapter 13.9**.

13.4.8 Chip Select Lines, $\overline{CSx}$

The EBU provides four user chip selects, $\overline{CS0}$, $\overline{CS1}$, $\overline{CS2}$ and $\overline{CS3}$. The address ranges for which these chip selects are generated for FBU based controlled transfers are programmed via the address select registers, EBU_ADDSELx, in a very flexible way (see **Section 13.7.1**).

The BIFU uses only $\overline{CS0}$ for all BIFU initiated transfers. Chip Select line $\overline{CS0}$ is also the default chip select used automatically by the FBU for external boot after reset. See **Section 13.5** for details.

If overlapping address regions are programmed in the EBU_ADDSELx registers, only one chip select — the one with the lower number (higher priority) — will be activated on an access within the overlapping address range.

If the number of chip select lines is not sufficient, additional chip-select signals can be generated by combining one chip select output with some address bits. In this case, all generated chip selects must share the same EBU timing and data width parameters. **Figure 13-3** shows how $\overline{CS3}$ can be divided into four smaller regions. Using this solution, the regions must be of equal size.

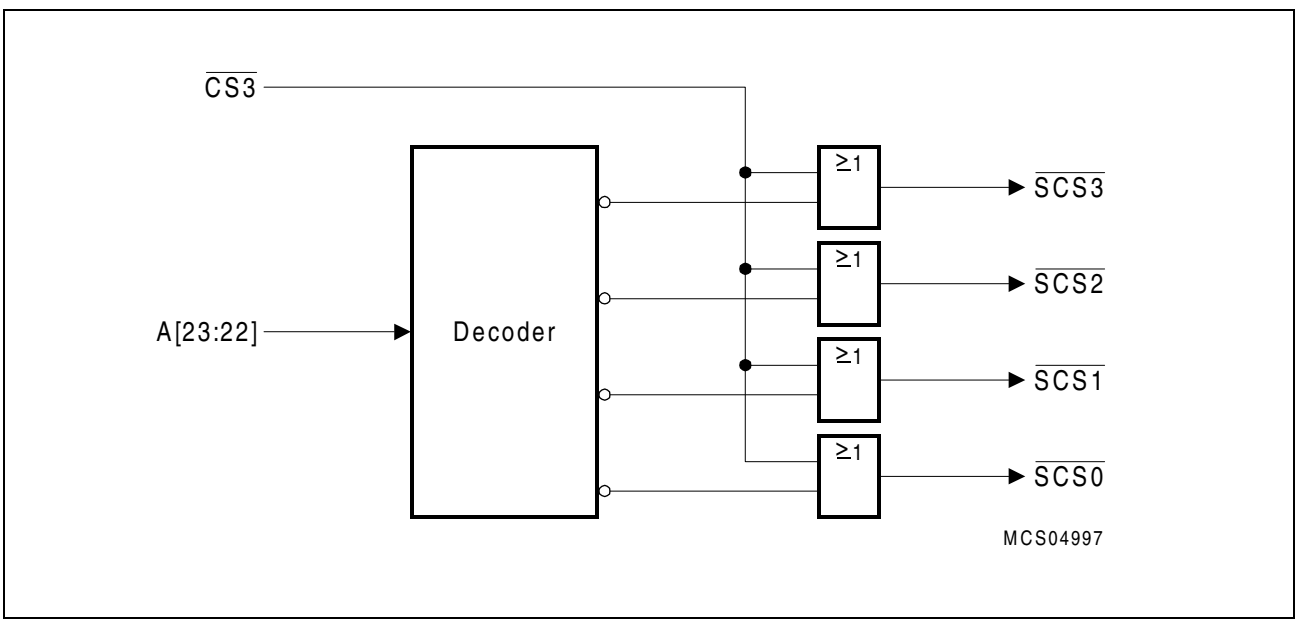


Figure 13-3 Simple Chip Select Expansion

13.4.9 Instruction Fetch Indication Signal, $\overline{CODE}$

To be able to distinguish instruction fetch accesses from data load/store accesses on the external bus, the EBU provides the signal $\overline{CODE}$. A low level on this line indicates an instruction fetch of the CPU, performed via the Program Memory Unit, PMU (regardless if the BIFU or the FBU is used for the transfer). Accesses to the external bus by any other internal FPI Bus master (such as the DMU or the DMA), are indicated through a high level of signal $\overline{CODE}$.

13.4.10 Emulation/Instrumentation Support Signal $\overline{CSEMU}/\overline{CSOVL}$

To support emulation, debugging and instrumentation, the EBU provides a special chip select, $\overline{CSEMU}/\overline{CSOVL}$. A detailed description of this signal can be found in **Section 13.8**.

13.5 EBU Boot Process

If external boot is selected — meaning that after reset, initial code execution begins from external memory — the EBU needs to access the external default or boot memory. However, because no application software has been executed yet, there is no information inside the EBU concerning the type of external memory and the proper access parameters to it.

To get around this problem, the EBU first starts a “blind” boot access to the external memory using a set of default values. This boot access is designed such that the EBU can access the external boot memory without knowing the exact parameters of it. In this way, the EBU retrieves additional detailed access information from a predefined location in the boot memory. It configures itself according to these parameters, and then performs the first true code fetch from location 0 of the boot memory, now with the proper access parameters. **Figure 13-4** gives an overview of this boot access.

Naturally, this boot access can handle only a limited variety of external boot memory types and access schemes. The boot memory must be either a ROM, EPROM, Flash-EPROM or a RAM memory compatible to these types. The access is in demultiplexed mode only. The memory can be 8-, 16-, or 32-bit wide.

The boot memory must be connected to $\overline{CS0}$, the EBU uses registers EBU_ADDSEL0 and EBU_BUSCON0 for the access. Its reset values are such that the boot access is performed in the following way:

- EBU is external bus master
- $\overline{CS0}$ is activated (low)
- Addresses A[23:0] are driven to $00\,0004_H$ (accessing offset address 4_H in the memory); demultiplexed address mode
- $\overline{RD}$ is activated (low)
- RD/$\overline{WR}$ is deactivated (high)
- Data lines D[31:0] are switched to input
- Byte control signals $\overline{BC}$[3:0] are in control mode and active (low)
- Signal $\overline{CODE}$ is activated (low)
- A number of wait-states will be inserted in the read access (EBU_BUSCON.WAITRDC $\times$ EBU_BUSCON0.CMULTR = 48 $\times$ 1)
- Evaluation of the $\overline{WAIT}$/$\overline{IND}$ input is disabled

With these default parameters, the EBU is able to activate a boot memory that is one of the types listed above, and is able to read the value stored at location 04_H of the memory. The information stored there is used to adjust the access parameters such that they exactly fit to the boot memory. This boot information needs to be programmed as shown in the following diagram.

BOOTCFG
EBU External Boot Memory Configuration Word

Boot Memory Offset Address + 04_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CFG 32	CMULT CMULTR		BCGEN		WAITRDC [4:3]		WAIT INV	CFG 16	SET UP	WAIT		WAITRDC [6:5]		–	–

Note: The gray shaded bit fields are only evaluated by the EBU if bit field BOOTCFG.7 (bit CFG16) is set.

Bit fields and bits WAITRDC, WAIT, SETUP, WAITINV, BCGEN, CMULTR, and CMULT have exactly the same meaning as the respective fields in register EBU_BUSCON0 (see **Section 13.11.4**). At the end of the boot access, the EBU_BUSCON0 fields will be overwritten with the respective values read from the configuration word BOOTCFG.

Note: During boot, the upper two bits (8-bit boot memory data width selected) or the upper four bits (16-/32-bit boot memory data width selected) of bit field WAITRDC of $\overline{CS0}$ will be loaded.

Note: During boot, bit fields CMULTR and CMULT are loaded with the same value defined by the external boot memory configuration word BOOTCFG[14:13] when 16/32-bit boot memory data width is selected.

Bits CFG16 and CFG32 determine the data width of the boot memory according to the **Table 13-4**:

Table 13-4 Boot Memory Data Width Encoding

CFG32	CFG16	Boot Memory Data Width
don't care	0	00_B: 8-bit
0	1	01_B: 16-bit
1	1	10_B: 32-bit

If the data width of the boot memory is 8-bit, bit CFG16 is set to 0, and the EBU will only read and evaluate bits 7..0 of the boot word BOOTCFG. The parameters for BCGEN, CMULTR and CMULT then must be set by software. It is advisable to perform this early in the initialization code of the application because the EBU uses the default number of wait states to read the boot memory.

If the data width of the boot memory is 16-bit or 32-bit, bit CFG16 in the boot word is set to 1, and the EBU reads and evaluates bits [15:0] of the boot word BOOTCFG. Bit CFG32 is set to 0 in case of a 16-bit boot memory, and set to 1 if the data width of the boot memory is 32-bit. The EBU will update bit field EBU_BUSCON0.PORTW according to the data width of the boot memory.

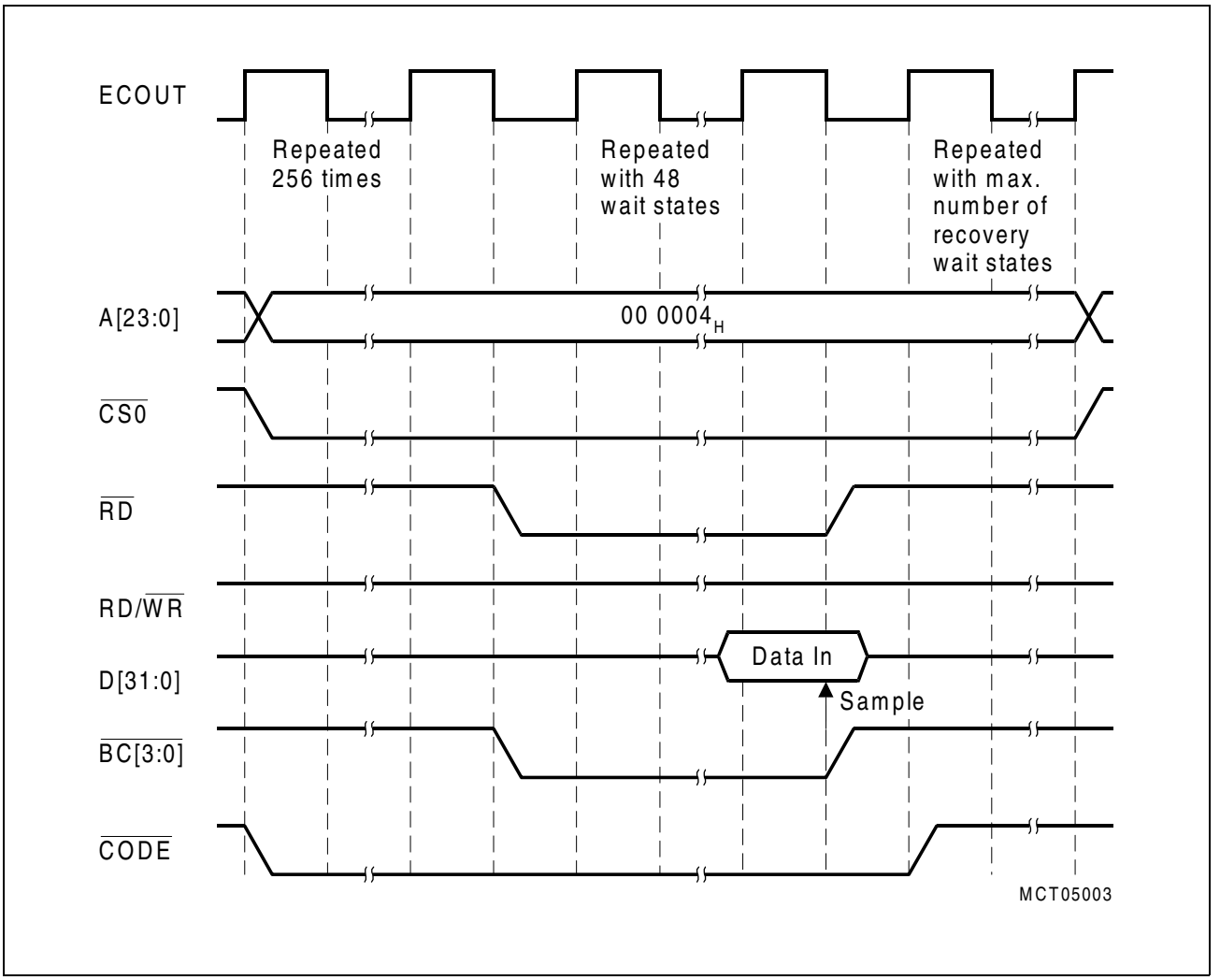


Figure 13-4 EBU Boot Process after Reset

At the beginning of this sequence, 256 clock cycles will be inserted to satisfy the recovery needs of external synchronous devices like flash ROMs between the time that reset becomes inactive and the above-described access occurs.

Note: After a boot operation of the TC1765 with combinations of CFG[2:0] except CFG[2:0] = 10X$_B$ the EBU is disabled and must be enabled by setting SCU_CON.EBUEN before it can be used.

13.6 FBU Operation for FPI Bus Based Transfers

The following subsections provides the detailed description for transactions initiated by FPI bus based transfer requests.

13.7 FBU Memory Map

When an internal FPI Bus master wants to perform a read or write transaction from/to a device connected to the external bus, it sends out the address onto the FPI Bus. The address needs to be in the address ranges defined as external, as shown in **Table 13-5**.

Table 13-5 EBU External Address Ranges

Segment	Address Range	Description
10	A000 0000_H - AFFF $FFFF_H$	External memory space (cached area)
11	B000 0000_H - BDFF $FFFF_H$	External memory space (non-cached area)
	BE00 0000_H - BEFF $FFFF_H$	External emulator memory (non-cached area)
14	E000 0000_H - EFFF $FFFF_H$	External peripheral and data memory space (non-cached area)

The EBU reacts to addresses in these ranges only. It compares the address sent from the FPI Bus master against the address ranges pre-programmed in its address select registers, EBU_ADDSELx. If it finds a match in one (or more) of the address regions, it selects the associated bus control register, EBU_BUSCONx, for that region, and starts to perform the external access according to the parameters programmed in the EBU_BUSCONx register.

On a write operation, the write data from the FPI Bus master is stored inside the EBU, and the master can continue with its other tasks. The EBU will take care of properly storing the data to the external device.

On a read operation, the FPI Bus master has to wait until the EBU has retrieved the data from the external device and has sent it to the master via the FPI Bus. The internal FPI Bus is blocked for that time, no other transaction can take place.

13.7.1 FBU External Address Regions ($\overline{CSx}$)

The FBU provides five programmable address regions (including the emulator range), each with its own chip select. The access parameters for each of the region can be programmed individually to accommodate different types of external devices. Four of these regions are provided for normal user application purposes, while the fifth one is reserved for emulator usage.

Two EBU registers and a chip select line are dedicated to each of the regions. The address range of the region is programmed through the address select register, EBU_ADDSELx (x = 3..0). The access parameters for the external device in that region

are programmed through the respective bus control register, EBU_BUSCONx. The access to the external device is performed using the associated chip select line, $\overline{CSx}$. **Table 13-6** summarizes the registers and chip selects associated with the five regions.

Table 13-6 FBU Address Regions, Registers and Chip Selects

Address Region	Address Select Register	Bus Control Register	Chip Select
User region 0	EBU_ADDSEL0	EBU_BUSCON0	$\overline{CS0}$
User region 1	EBU_ADDSEL1	EBU_BUSCON1	$\overline{CS1}$
User region 2	EBU_ADDSEL2	EBU_BUSCON2	$\overline{CS2}$
User region 3	EBU_ADDSEL3	EBU_BUSCON3	$\overline{CS3}$
Emulator region	EBU_EMUAS	EBU_EMUBC	$\overline{CSEMU}$/$\overline{CSOVL}$

13.7.1.1 Address Region Selection

Any FPI Bus address belonging to one of the external ranges shown in **Table 13-5** activates the FBU (provided the FBU is idle). It picks up the address and compares it to the five address regions programmed through its address select registers (including the emulator range). Each address select register (EBU_ADDSELx, EBU_EMUAS) contains four bit fields (see also **Section 13.11.3** and **Section 13.11.7**):

- Bit REGEN is the enable control of that region. If the region is disabled (REGEN = 0), any address in that region presented to the FBU will result in a bus error reported back to the master requesting the access and to the FPI Bus Control Unit, BCU (which in turn will generate an interrupt request, if the BCU is configured for). Also the chip select associated with that range is disabled.
- Bit field BASE specifies address bits A[31:12] of region x, where A[31:28] must only point to segments 10, 11 and 14 which are covered by the FBU (see **Table 13-5**).
- Bit field MASK specifies how many bits of an FPI Bus address must match the contents of the BASE(x) bit field (to a maximum of 15, starting with A[26]). (Note that address bits A[31:27] must always match.) This parameter defines the length of a region.
- The MIRRORE bit allows access of the same physical external device in two different segments. One of these segments is programmable through the BASE bit field, while the other is fixed to be Segment 11 (1011_B). Setting bit MIRRORE defines a second address region in Segment 11 with the same size and intra-segment start address as the main region. An address in Segment 11 with the same intra-segment offset address (A[27:0]) as defined for the main region will activate that region.

Figure 13-5 illustrates how the comparison of the FPI Bus address to the address region setup in register EBU_ADDSELx/EBU_EMUAS is performed to determine whether or not a region is selected.

Note: The address region selection logic always uses all 32 address bits for decoding while only A[23:0] are output at pins.

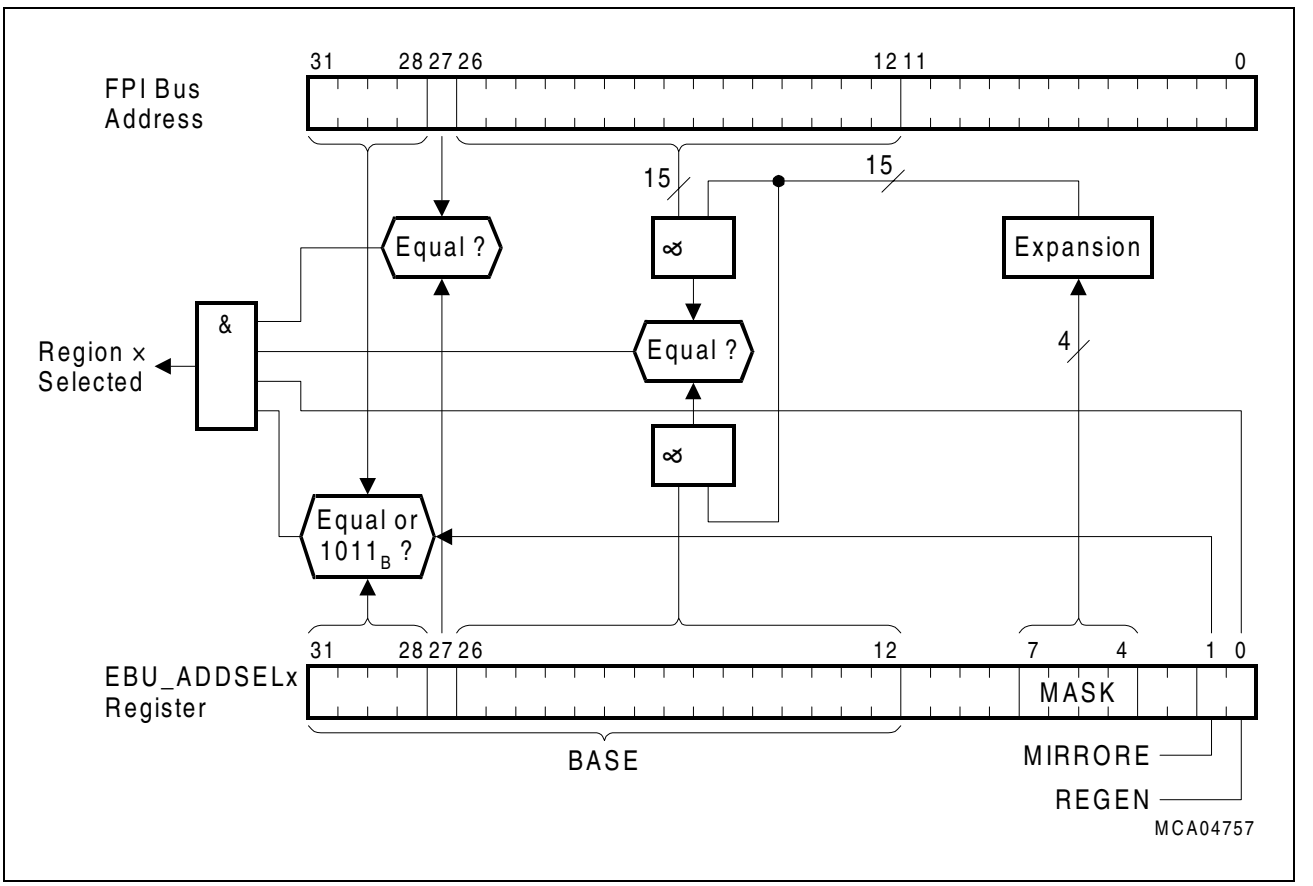


Figure 13-5 Address Region Selection

This address region scheme described above implies the following:

- The smallest possible address region is 2^{12} bytes (4 KBytes)
- The largest possible address region is 2^{27} bytes (128 MBytes)
- The start address of a region depends on the size of the region. It must be at an address which is a multiple of the size of a region; for example, the smallest region can be placed on any 4-KByte boundary, while the largest region can be placed on 8 MBytes boundaries only.

Table 13-7 shows the possible region sizes and start granularity, as determined by the programming of the MASK bit field. The range of the offset address within such a region is also given.

Table 13-7 FBU Address Regions Size and Start Address Relations

MASK	No. of Address Bits Compared to BASE[26:12]	Range of Address Bits Compared to BASE[26:12]	Region Size and Start Address Granularity	Range of Offset Address Bits within Region
1111_B	15	A[26:12]	4 KBytes	A[11:0]
1110_B	14	A[26:13]	8 KBytes	A[12:0]
1101_B	13	A[26:14]	16 KBytes	A[13:0]
1100_B	12	A[26:15]	32 KBytes	A[14:0]
1011_B	11	A[26:16]	64 KBytes	A[15:0]
1010_B	10	A[26:17]	128 KBytes	A[16:0]
1001_B	9	A[26:18]	256 KBytes	A[17:0]
1000_B	8	A[26:19]	512 KBytes	A[18:0]
0111_B	7	A[26:20]	1 MByte	A[19:0]
0110_B	6	A[26:21]	2 MBytes	A[20:0]
0101_B	5	A[26:22]	4 MBytes	A[21:0]
0100_B	4	A[26:23]	8 MBytes	A[22:0]
0011_B	3	A[26:24]	16 MBytes	A[23:0]
0010_B	2	A[26:25]	32 MBytes[1)]	A[24:0]
0001_B	1	A[26]	64 MBytes[1)]	A[25:0]
0000_B	0	–	128 MBytes[1)]	A[26:0]

1) These region size selections do not affect the external address bus of the TC1765 because A24, A25 and A26 are not output at pins (only A[23:0] are available at TC1765 pins).

Due to the scheme shown in **Table 13-7**, memory regions can overlap and there can be gaps between regions. FBU actions in these cases are as follows.

1. An address lies in exactly one defined region:
   The FBU will perform the requested access to external memory.
2. An address lies in more than one region (overlapping regions):
   The access is performed to the region with higher priority where region 0 has the highest priority, region 3 has the lowest.
3. The address does not lie in any region, or lies in a disabled region:
   In case of an unknown external address or disabled region, the FBU will return an error-acknowledge code on the FPI Bus.

The mirror function of the FBU, selected through the MIRRORE bit is especially useful if, for instance, cached and non-cached data or code accesses to the same external memory need to be performed. The distinction between cached and non-cached accesses is made via the address: Segment 10 addresses are always cached, while Segment 11 addresses are never cached. By mirroring an external device into both Segments 10 and 11, the same location in the device can be accessed via an address in Segment 10, resulting in a cached access, or via the same offset address in Segment 11, resulting in an non-cached access. The respective address select register for the external device is programmed such that a Segment 10 address region is defined, and the MIRRORE bit is set. Setting the MIRRORE bit results in that a second address region in Segment 11 is defined, with the same intra-segment start address and size as the one in Segment 10. An access to one of these two regions results in the activation of the EBU_BUSCONx register and $\overline{\text{CSx}}$ signal associated with that EBU_ADDSELx register, thus accessing the same physical external device regardless whether the effective address is in Segment 10 or Segment 11.

13.7.1.2 Address Region Parameters

When a FPI Bus address presented to the FBU is found to belong to one of its programmed active (enabled) address regions, the FBU performs the external bus access according to the global FBU parameters stored in register EBU_CON and according to the individual parameters stored in the bus control register, EBU_BUSCONx, associated with that address region.

The bit fields in EBU_BUSCONx (see **Section 13.11.4**) control the following parameters of the access as shown in **Table 13-8**.

Table 13-8 Programmable Characteristics of an Address Region

Bit Field	Description
WRITE	Write protection enable/disable
SETUP	Address setup time
WAIT	$\overline{\text{WAIT}}$/$\overline{\text{IND}}$ input enable/disable
WAITINV	$\overline{\text{WAIT}}$/$\overline{\text{IND}}$ input active level control
PORTW	Data width of external device
ALEC	ALE length control
BCGEN	Byte control signal timing characteristics
WAITWRC	Number of wait states for write access
WAITRDC	Number of wait states for read access
HOLDC	Number of wait states for write data hold
RECOVC	Number of wait states for recovery cycle
CMULT	Multiplier for number of wait states
CMULTR	Multiplier for number of read cycles

The FBU reads these parameters from the EBU_BUSCONx register associated with the selected address range and performs the access accordingly. The chip select line $\overline{\text{CS}}$x associated with that range is activated during the access.

13.7.2 Driver Turn-Around Wait States

Besides the wait states that can be inserted into an external access, the FBU supports the insertion of wait states in between consecutive accesses. This may be necessary if the current access is to a different address region than the previous one, or if a read access is followed by a write access, or vice versa. The insertion of wait states between the accesses allows the timing of accesses to external devices to be fine-tuned to gain higher performance.

When, for instance, a number of read accesses to an external memory are performed with a demultiplexed bus configuration, the memory is the only driver on the data bus, providing its data onto the bus. The memory is constantly selected via its chip select. If an access to a different device is performed (different address region, different chip select), the memory is deselected, and the next device is selected. However, many memory devices need a specific time to fully release the bus, to tri-state their output drivers. Recovery wait states would need to be inserted at the end of the last access to the memory to ensure enough time to get off the bus before the next access occurs.

A similar situation is true if a read access is followed by a write access. The data bus driver role must change from the memory to the FBU. Again, the memory needs time to release the data bus, and recovery wait states need to be inserted.

If this recovery wait state insertion would be programmed via the address region parameters (see **Section 13.7.1.2**), the wait states would apply to every access to the device, thus, slowing down the access performance. Instead, the FBU offers the option to insert such wait states either between accesses to different address regions (different chip selects) or between read and write accesses.

Programming of these wait states is done through register EBU_CON. Between 0 and 7 idle cycles can be inserted between accesses to different address regions via bit field DTACS, while DTARW provides the option to insert between 0 and 7 idle cycles between a read and a write access, or vice versa. With these options, access performance to external devices is significantly improved, especially when a number of consecutive access of the same type (read or write) is performed.

13.7.3 Data Width of External Devices

The FBU supports external devices with a data width of 8-, 16- or 32-bit. If the data width of an access is less than or equal to the width of the external device, no special conditions occur, and the FBU indicates the width of the data via the byte control lines $\overline{BC}[3:0]$. However, if the data width of an access is larger than the width of the external device, the data needs to assembled/disassembled. **Table 13-9** shows the FBU data assemble/disassemble operation.

Table 13-9 Data Assembly/Disassembly

FPI Bus Access Width	Data Width of External Device	FBU Operation for Demultiplexed Access	$\overline{BC3}$	$\overline{BC2}$	$\overline{BC1}$	$\overline{BC0}$
8-bit	8-bit	One byte access	high	high	high	low
	16-bit	One byte access on byte lane 0 (A[0] = 0)	high	high	high	low
		One byte access on byte lane 1 (A[0] = 1)	high	high	low	high
	32-bit	One byte access on byte lane 0 (A[1:0] = 00_B)	high	high	high	low
		One byte access on byte lane 1 (A[1:0] = 01_B)	high	high	low	high
		One byte access on byte lane 2 (A[1:0] = 10_B)	high	low	high	high
		One byte access on byte lane 3 (A[1:0] = 11_B)	low	high	high	high
16-bit	8-bit	first Byte access with A[0] = 0	high	high	high	low
		second Byte access with A[0] = 1 [1)]	high	high	low	high
	16-bit	Halfword access on byte lanes 0 and 1	high	high	low	low
	32-bit	Halfword access on byte lanes 0 and 1	high	high	low	low
		Halfword access on byte lanes 2 and 3	low	low	high	high

Table 13-9 Data Assembly/Disassembly (cont'd)

FPI Bus Access Width	Data Width of External Device	FBU Operation for Demultiplexed Access	$\overline{BC3}$	$\overline{BC2}$	$\overline{BC1}$	$\overline{BC0}$
32-bit	8-bit	first Byte access with A[1:0] = 00_B	high	high	high	low
		second Byte access with[1] A[1:0] = 01_B	high	high	low	high
		third Byte access with[1] A[1:0] = 10_B	high	low	high	high
		forth Byte access with[1] A[1:0] = 11_B	low	high	high	high
	16-bit	first Halfword access on byte lanes 0 and 1	high	high	low	low
		second Halfword access on[2] byte lanes 2 and 3	low	low	high	high
	32-bit	One word access on byte lanes 0..3	low	low	low	low

1) This byte access is performed automatically in consecutive to the previous byte access.

2) This half-word access is performed automatically in consecutive to the pervious halfword access.

13.7.4 Basic Access Timing

This section describes the basic access sequences of the FBU to external devices. Refer to the TC1765 Data Sheet for detailed timing diagrams and timing values.

Note: All timings described in this section are specified relative to the ECOUT signal.

Access to Demultiplexed Devices

Devices with demultiplexed access can be controlled by separate $\overline{RD}$ and $\overline{WR}$ signals (Intel™-style). **Figure 13-6** shows the basic sequence of a read access in demultiplexed mode. The sequence is detailed in the following:

- Phase 0: This is an optional cycle, providing a longer address and chip select setup time before the read signal. This cycle is enabled through bit SETUP in register EBU_BUSCONx.
- Phase 1a: This cycle is always part of a read access. Please note that the read signal is activated (low) on the falling edge of ECOUT.
- Phase 1b: This is an optional cycle that can be repeated several times. Enabling and programming of the length of this phase is performed through bit fields WAITRDC and CMULTR in register EBU_BUSCONx. The number of clock cycles is determined through WAITRDC $\times$ CMULTR. The minimum number of wait states is 0, the maximum is 127 $\times$ 8.
- Phase 2: This cycle is always part of a read access. Please note that the read signal is deactivated (high) on the falling edge of CLKOUT. The data input is sampled and latched with this clock edge (EBU_BUSCONx.ES = 0) or with the rising edge of CLKIN when early sample is enabled with EBU_BUSCONx.ES = 1.
- Phase 3: This is an optional cycle that can be repeated several times. Some devices may require this recovery cycle before they can be accessed again. Enabling and programming of the length of this phase is performed through bit field RECOVC in register EBU_BUSCONx. The number of clock cycles can be 0..3.

The minimum time required for a read access in demultiplexed mode is two clock cycles (phases 1a and 2). The maximum time is 1022 clock cycles (Phase 0: one clock; Phase 1a: one clock; Phase 1b: 127 $\times$ 8 clocks; Phase 2: one clock; Phase 3: 3 clocks).

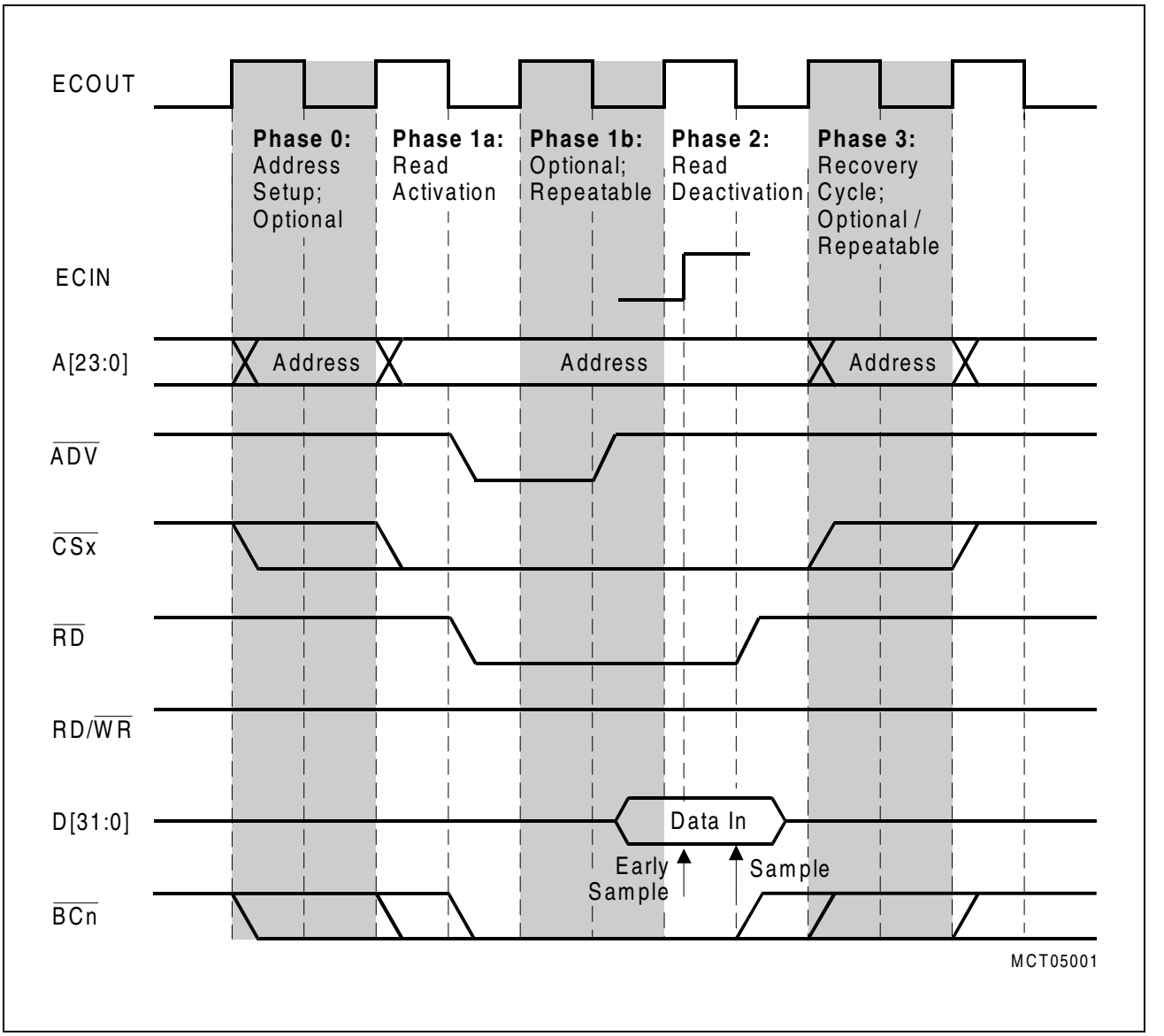


Figure 13-6 Basic Read Access Timing in Demultiplexed Mode

Note: The timing of the byte control signals BCn can be programmed to be like that of the control signals RD, RD/WR, or chip-select. (Having RD/WR only is equivalent to a byte-read enable function).

Figure 13-7 shows the sequence for a write access, detailed as follows.

- Phase 0: This is an optional cycle, providing a longer address and chip select setup time before the read signal. This cycle is enabled through bit SETUP in register EBU_BUSCONx.
- Phase 1a: This cycle is always part of a write access. Please note that the write signal is activated (low) on the falling edge of ECOUT. The output of the write data starts in this cycle.
- Phase 1b: This is an optional cycle that can be repeated several times. Enabling and programming of the length of this phase is performed through bit fields WAITWRC and

CMULT in register EBU_BUSCONx. The number of clock cycles is determined through WAITWRC × CMULT. The minimum number of clock cycles is 0, the maximum is 16 × 7.

- Phase 2: This cycle is always part of a read access. Please note that the write signal is deactivated (high) on the falling edge of ECOUT. The output data is held stable for at least another 1/2 clock cycle.
- Phase 3: This is an optional cycle, which can be repeated several times. The output data is held stable by the FBU during this phase to allow extra time for the external device to process the write data. Enabling and programming of the length of this phase is performed through bit field HOLDC in register EBU_BUSCONx. The number of clock cycles can be 0..3.
- Phase 4: This is an optional cycle that can be repeated several times. Some devices may require this recovery cycle before they can be accessed again. The output data is not driven anymore in this phase. Enabling and programming of the length of this phase is performed through bit field RECOVC in register EBU_BUSCONx. The number of clock cycles can be 0..3.

The minimum time required for a write access in demultiplexed mode is two clock cycles (phases 1a and 2). The maximum time is 121 clock cycles (Phase 0: one clock; Phase 1a: one clock; Phase 1b: 16 × 7 clocks; Phase 2: one clock; Phase 3: 3 clocks; Phase 4: 3 clocks).

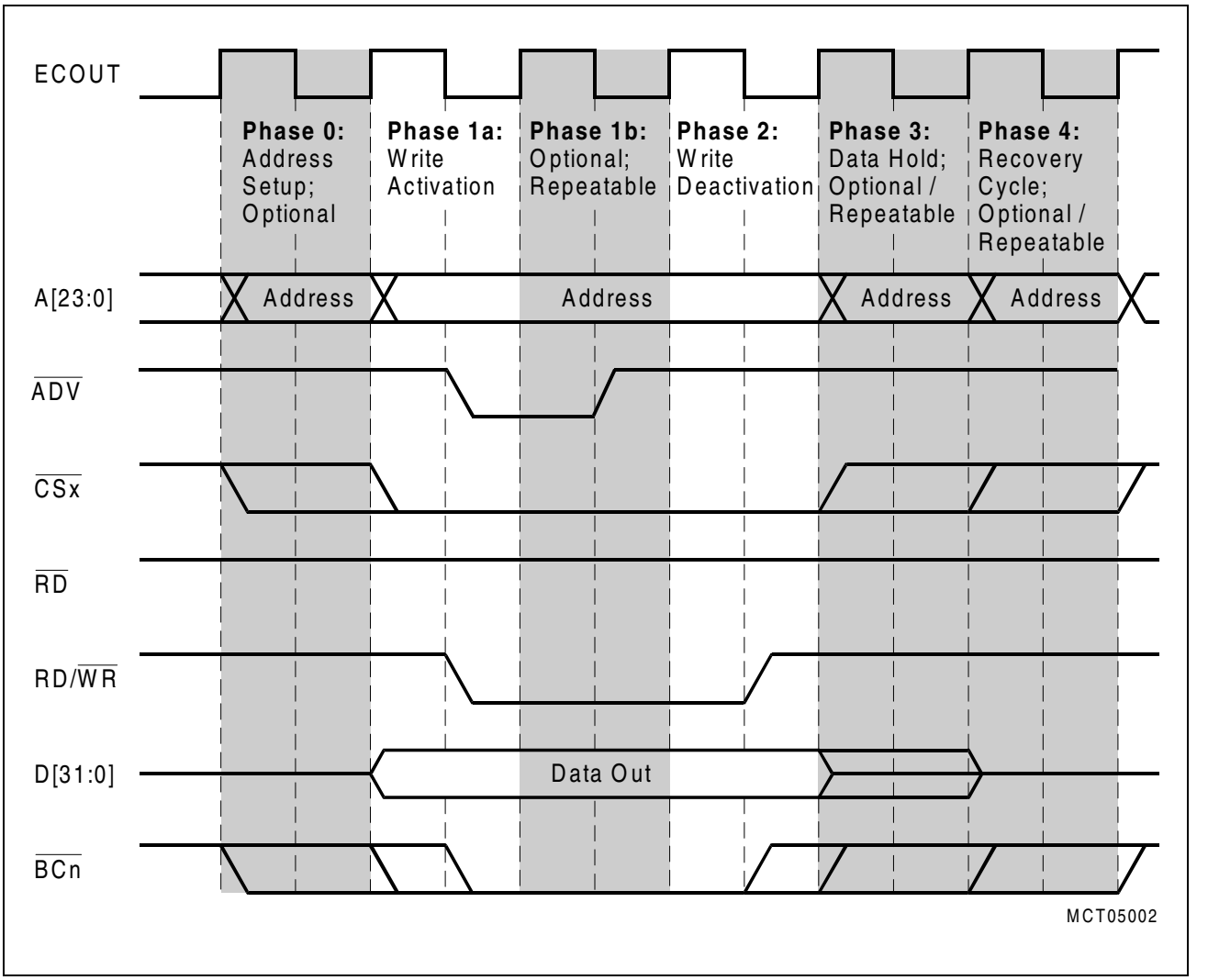


Figure 13-7 Basic Write Access Timing in Demultiplexed Mode

Note: The timing of the byte control signals BCn can be programmed to be like that of the control signals RD, RD/WR, or chip-select. (Having RD/WR only is equivalent to a byte-write enable function).

13.7.5 Variable Wait State Control, Burst Abort $\overline{\text{WAIT}}/\overline{\text{IND}}$

This is an input signal to the FBU allowing the external device to force the FBU to insert additional wait states into the access. The $\overline{\text{IND}}$ functionality of this input is not used by the FBU. $\overline{\text{WAIT}}/\overline{\text{IND}}$ can be enabled/disabled by software and programmed to be active low or active high (the active level forces additional wait states). Its sampling by the FBU can be selected to be synchronous or asynchronous to the ECOUT.

A fixed number of initial wait states should be programmed for the access because the external device usually requires time to react to an access and properly set $\overline{\text{WAIT}}$ to the appropriate level, and because the FBU requires time to sample and react to the $\overline{\text{WAIT}}$ signal. The FBU inserts the programmed number of wait states before evaluating the level of $\overline{\text{WAIT}}$.

If synchronous mode is selected, $\overline{\text{WAIT}}$ is sampled on the falling edge of ECOUT so that it can be evaluated at the next rising edge of the clock. In asynchronous mode, the $\overline{\text{WAIT}}$ signal needs to go through another register for synchronization, so that it is evaluated at the second rising clock edge. Thus, asynchronous operation of $\overline{\text{WAIT}}$ may results in one additional wait state compared to synchronous operation, if the signal was deactivated at the same time.

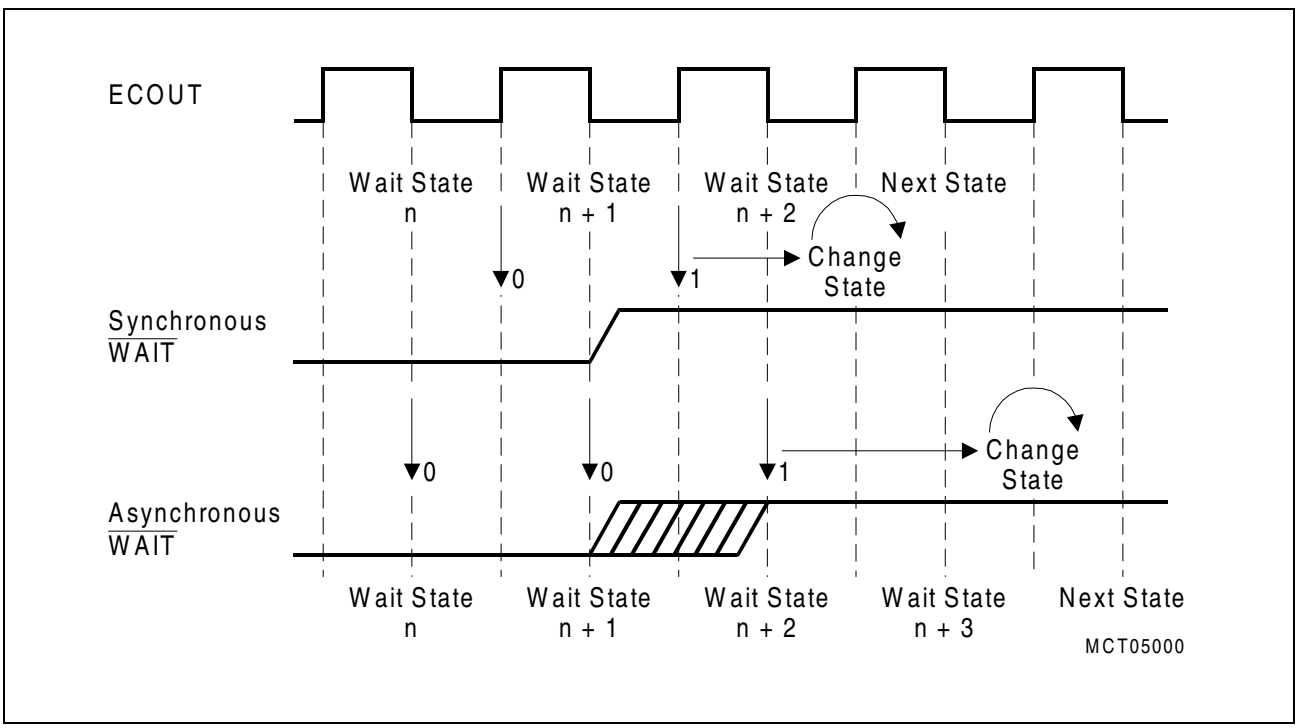


Figure 13-8 Sampling of the $\overline{\text{WAIT}}$ Signal

Note: Synchronous mode can be used if two TC1765 chips are connected and running with the same frequency. They must be using the same external clock source with their PLLs bypassed. Due to the time needed for the $\overline{\text{WAIT}}$ signal generation and synchronization, at least one fixed wait-state must be configured in register EBU_BUSCONx for the access to the second TC1765.

13.7.6 Execution of Transfer Requests to the FBU

There are special considerations when an internal FPI Bus master requests an external bus transaction from the FBU. Several conditions apply to determine the FBU's reaction: It first checks the address present on the FPI Bus against the address ranges programmed in its address select registers, EBU_ADDSEL*x*, as described in **Section 13.7.1**. If the address does not fall within one of the ranges, the FBU returns an error acknowledge code onto the FPI Bus.

If the address belongs to one (or more) of the programmed ranges, further reaction depends on the following conditions:

- If the FBU is currently busy with another transaction requested previously by another master, it issues a retry acknowledge code onto the FPI Bus to inform the requesting master that it needs to re-generate the request at a later point in time.
- If the FBU was idle and is currently already the owner (the master) of the external bus, it will start to perform the required access.

13.7.7 Atomic Read-Modify-Write Accesses

A read-modify-write access (RMW) consists of a read immediately followed by a write to the same location. Such an operation is used, for instance, to modify a semaphore bit. There must be no access of another device between, that is, the operation must be atomic. Otherwise, a deadlock situation could occur if the other device accesses the same location before the first one has finished its modification. Special provisions are implemented in the FBU to support RMW operations on the external bus.

13.8 Emulation Support

The TC1765 supports emulation and debugging via a number of measures. Some of features are provided through the EBU via the external bus. A special emulation boot is provided after reset which activates the EBU to direct code and data accesses from the CPU to a dedicated emulator memory region. Additionally, accesses to application memory can be redirected to emulation memory during debugging and emulation to allow replacement of application memory contents with special emulation memory.

Note: The EBU uses special registers and signals for this emulation support. Using these resources for normal application purposes may result in conflicts when using emulators/debuggers, and may severely hinder proper debugging. It is strongly recommended to exclude this signal from system design unless it is ensured that the tool environment used does not need this signal.

13.8.1 Emulation Boot and Emulation Memory

One of the boot options of the TC1765, selectable during reset, is to start execution out of a special external emulation memory. This memory is connected to the external bus of the EBU in a standard way, however, a special chip select, $\overline{\text{CSEMU}}$, is provided for this memory. The address range for this emulation memory is predefined to Segment 11, starting at address BE00 0000_H with a size of 16 MBytes.

If emulation boot is selected during reset, then after the end of the reset sequence the Program Counter (PC) of the CPU is set to BE00 0000_H (pointing to the emulation memory) and the EBU is enabled. The EBU has an address select register, EBU_EMUAS, and a bus control register, EBU_EMUBC, dedicated to the emulation memory. The address area and access parameters in these registers are set to predefined default values for a certain type of emulation memory. Thus, the EBU does not need to perform a boot access to the memory to retrieve further configuration data, as required for a normal boot. The code fetch requests from the CPU activate the EBU, which in turn performs a respective access to the emulation memory.

In this way, emulation software instead of application software is executed directly after reset. After having performed necessary initialization and programming, the emulation software usually executes a soft reset with the proper boot configuration to perform a normal boot and returning to the application software.

13.8.2 Overlay Memory

During emulation and debugging, it is often necessary to modify or replace the application code. While this is not very difficult to do with easily writable memories, such as RAMs, it can be awkward or even not possible without removing the memory or adding special provisions for on-board reprogramming when the code is stored in non-volatile memory such as a ROM or an EPROM.

The solution to this problem provided by the EBU is an overlay memory chip select, CSEMU/CSOVL. This chip select line can be programmed to be active in addition to the normal chip select connected to the application memory. An additional overlay memory can then be connected to the external bus, using this overlay chip select to activate it. Additionally, the CSEMU/CSOVL line is used to gate the read and write signals to the application memory. **Figure 13-9** gives an overview for such a configuration. Only the signals relevant for this feature are shown.

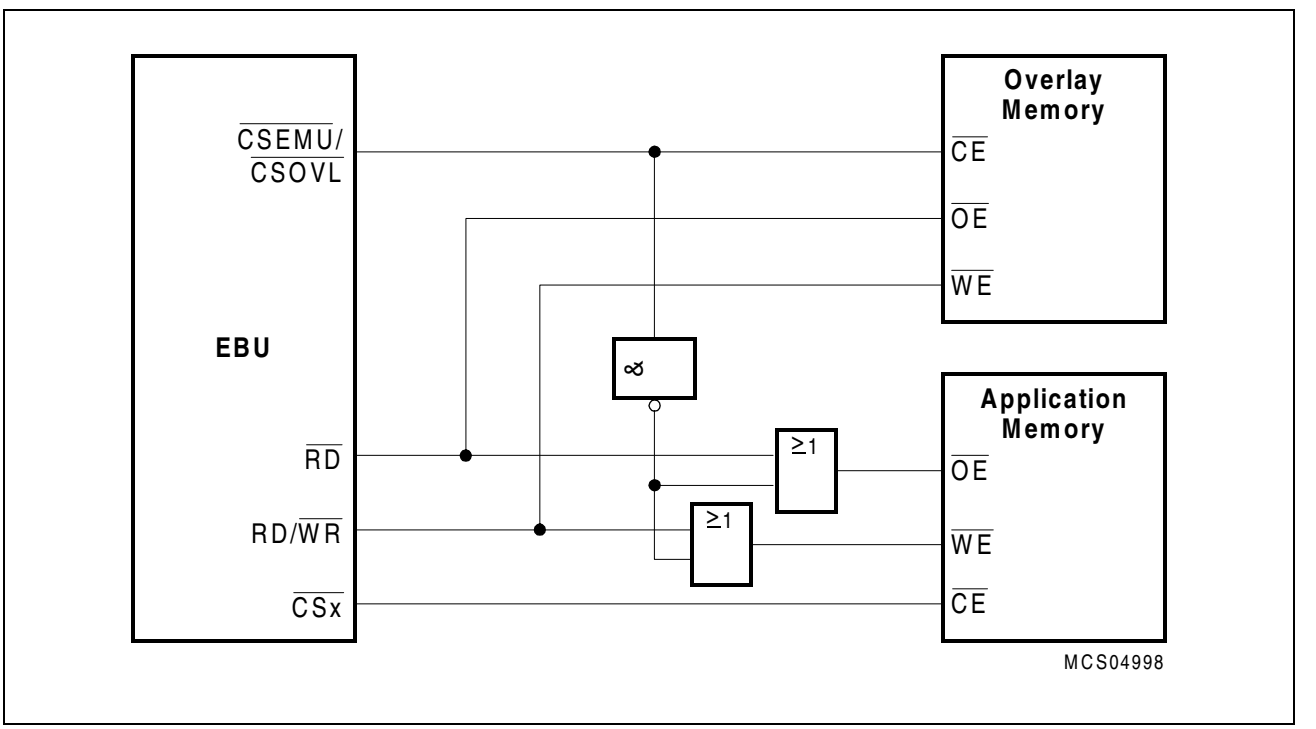


Figure 13-9 Use of the Overlay Chip Select

If the overlay chip select option is selected for an address range, then it is activated for all accesses to this address range in addition to the activation of the regular chip select, CSx. Thus, the overlay memory is activated for these accesses. To ensure that the regular application memory is not driving the bus on a read or storing the data on a write, the inverted overlay chip select controls two OR-gates to disable the read and write signals to the memory. In this way, the overlay memory is accessed instead of the application memory.

The selection of the overlay chip select is performed through register EBU_EMUCON. For each of the four regular chip selects CS[3:0], an enable bit for CSOVL is provided. It is possible to activate CSOVL for one or more of the regular chip selects.

Note: To guarantee proper access, the overlay memory must meet the same access requirements as the application memory. The access to it is performed according to the parameters programmed for the application memory via the EBU_BUSCONx register associated with the regular chip select.

*Note: Use of the overlay chip select feature is intended for emulation support. The circuitry shown in **Figure 13-9** is usually provided on the emulator probe. It does not need to be included in the application circuitry.*

13.8.3 Usage of $\overline{\text{CSEMU}}$/$\overline{\text{CSOVL}}$

For both emulation support function only one chip select signal ($\overline{\text{CSEMU}}$/$\overline{\text{CSOVL}}$) is available. The signal is activated whenever at least one of the three possible conditions (Emulation Boot, match of EBU_EMUAS or $\overline{\text{CSOVL}}$ enabled for an address region) is matched. Therefore only one of these functions should be used in a system. However, by proper software design and control, both functions can be used alternatively in one system.

13.9 BIFU Operation for Burst Mode Instruction Fetches

This section describes the synchronous burst Flash memory accesses that are initiated and controlled by the PMU via the BIFU and which use the EBU lines for external access.

13.9.1 Basic Functions

The PMU is designed to perform burst mode cycles for an external code Flash memory. These burst mode cycles are executed via a separate instruction fetch bus as shown in **Figure 13-1**. In general, the burst mode cycle capability provides the following features:

- Fully synchronous timing with flexible programmable timing parameters
- 16- and 32-bit data bus width
- Support of Fast Boot Block Flash Memory (like INTEL 28F800F3 and 28F160F3)
- Support of Burst Mode Flash Memory (like AMD 29BL162)

Figure 13-10 shows the basic configuration of external burst Flash memory connections.

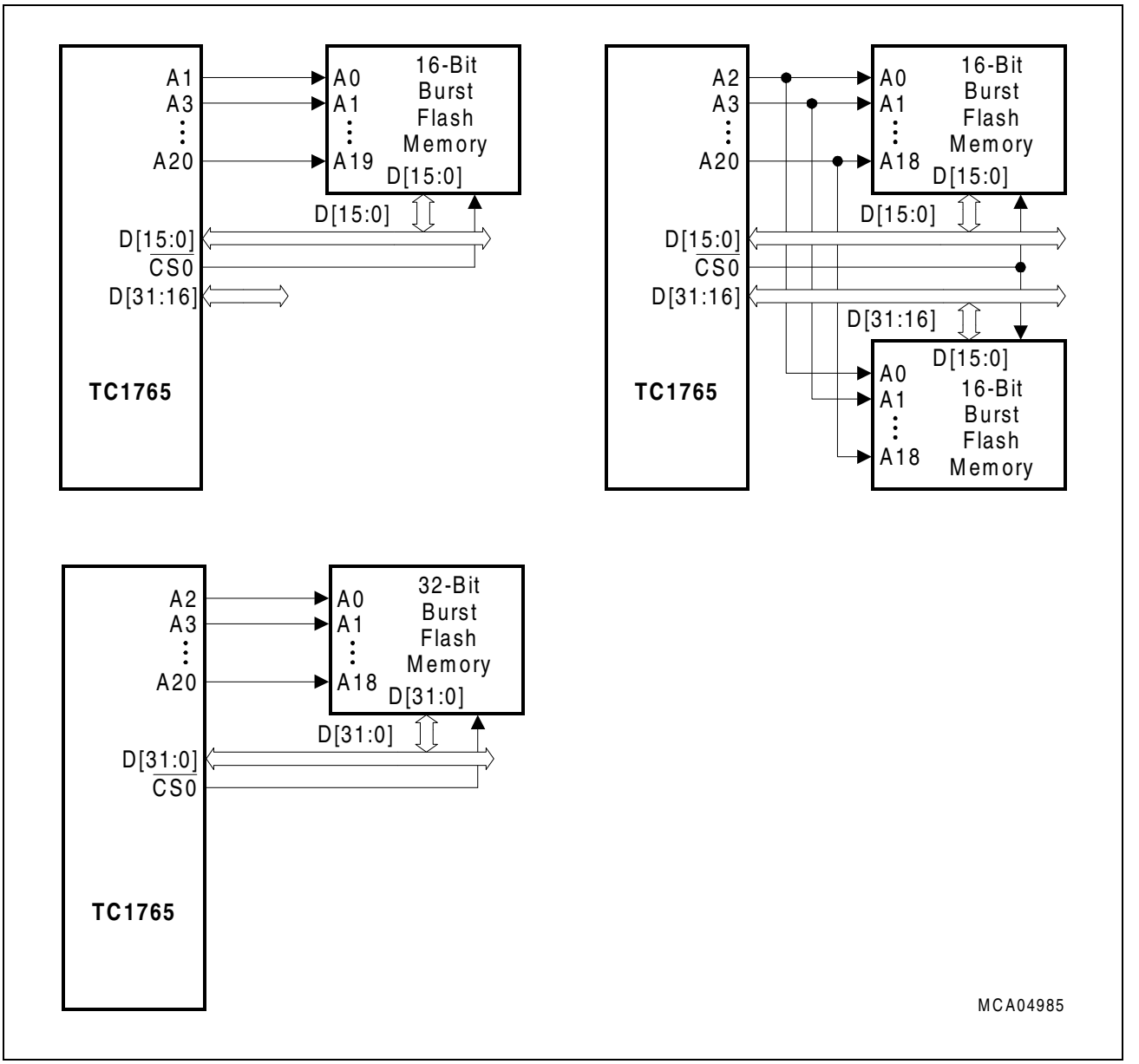


Figure 13-10 Basic Configurations of External Burst Flash Memory Connections

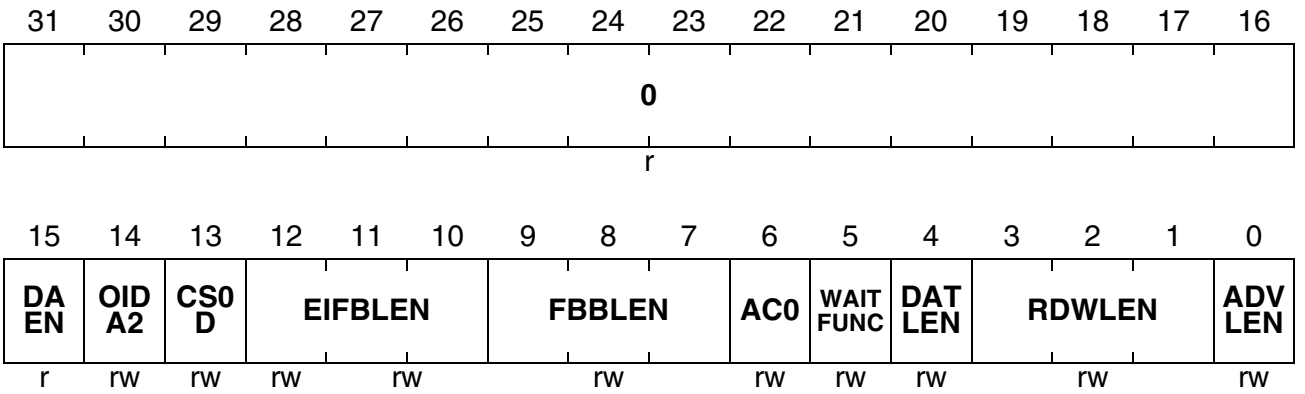
The data bus width for burst instruction fetches via the BIFU is defined by bit field EBU_BUSCON0.PORTW. PORTW is set e.g. after reset according the settings of CFG16 and CFG32 of the External Boot Memory Configuration Word BOOTCFG.

Note that within Segment 10 and partly 11 instruction fetches can be also executed via the FPI Bus using the asynchronous access schemes as described in **Section 13.6**. Two control bits in the SCU control register SCU_CON define which path in **Figure 13-1** is selected for instruction fetches (details on register SCU_CON see **Chapter 4**).

13.9.2 External Instruction Fetch Control Register

The external burst mode instruction fetches are controlled and defined by the PMU_EIFCON register which is located in the PMU.

PMU_EIFCON
PMU External Instruction Fetch Control Register **Reset Value: 0000 001F$_H$**


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0
r
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DA EN
OID A2
CS0 D
EIFBLEN
FBBLEN
AC0
WAIT FUNC
DAT LEN
RDWLEN
ADV LEN
r rw rw rw rw rw rw rw rw rw

Field	Bits	Type	Description
ADVLEN	0	rw	**Number of Address Cycles** This bit defines the number of address cycles. 0 One address cycle 1 Two address cycles (default after reset)
RDWLEN	[3:1]	rw	**Number of Read Wait Cycles** This bit field defines the delay between the address cycle and the initial data cycle (read wait cycles). 0_H-7_H 0 to 7 read wait cycles (7_H is default after reset)
DATLEN	4	rw	**Number of Data Cycles** This bit defines the number of data cycles. 0 One data cycle 1 Two data cycles (default after reset)
WAITFUNC	5	rw	**Operation of the $\overline{\text{WAIT}}$/$\overline{\text{IND}}$ Input** Defines the operation of the $\overline{\text{WAIT}}$/$\overline{\text{IND}}$ input. 0 The $\overline{\text{WAIT}}$/$\overline{\text{IND}}$ input operates as a wait data bus function on bursts (default after reset). 1 The $\overline{\text{WAIT}}$/$\overline{\text{IND}}$ input operates as a terminate burst function.
AC0	6	rw	**Reserved (Active 0)** This Bit is 0 after reset and can be read/written without any function. A 1 is read back if a 1 was written before.

Field	Bits	Type	Description
FBBLEN	[9:7]	rw	**Flash Burst Buffer Length** This bit field defines the maximum number of linear Flash burst data cycles which are supported by the Flash. If the PMU requests a burst which will cross an address boundary of a memory block defined by $n \times 2^{FBBLEN}$, the burst will terminate at this address boundary and a subsequent burst with the remaining will be initiated (new address cycle). 000_B Continuous mode (default after reset) 001_B 4 linear burst data cycles 010_B 8 linear burst data cycles 011_B 16 linear burst data cycles 100_B 32 linear burst data cycles 101_B 64 linear burst data cycles 110_B Reserved; don't use this combination 111_B Reserved; don't use this combination
EIFBLEN	[12:10]	rw	**External Instruction Flash Burst Length** This bit field defines the maximum number of burst data cycles which are initiated by the PMU. 000_B 1 word data access (default after reset) 001_B 2 word data accesses 010_B 4 word data accesses 011_B 8 word data accesses 100_B 16 word data accesses All other combinations are reserved and must not be used.
CS0D	13	rw	**Chip Select 0 Disable** This bit defines whether $\overline{CS0}$ is generated during burst mode accesses or not. 0 $\overline{CS0}$ is activated during code fetch (default after reset) 1 $\overline{CS0}$ is not activated during code fetch

Field	Bits	Type	Description
OIDA2	14	rw	**Overlay Initial Data Cycle to Address Cycle 2** This bit defines whether address cycle 2 and the initial data cycle are overlayed during an burst access if no read wait cycles are programmed. 0 No overlay performed 1 Address cycle 2 and initial data cycle are overlayed If read wait cycles are used, setting this bit will reduce the number of read wait cycles by one.
DAEN	15	rw	**Dynamic Address Enable** This bit defines whether the address during a burst stays stable at the starting during the whole burst sequence, or if the addresses for the subsequent data cycles is updated. 0 Address remains stable (default) 1 Address is updated during each data cycle This bit must not be set if wait states may be inserted via $\overline{\text{WAIT}}$/$\overline{\text{IND}}$ into the burst data cycles. In this case wrong data will be sampled
0	[31:16]	r	**Reserved**; read as 0; should be written with 0.

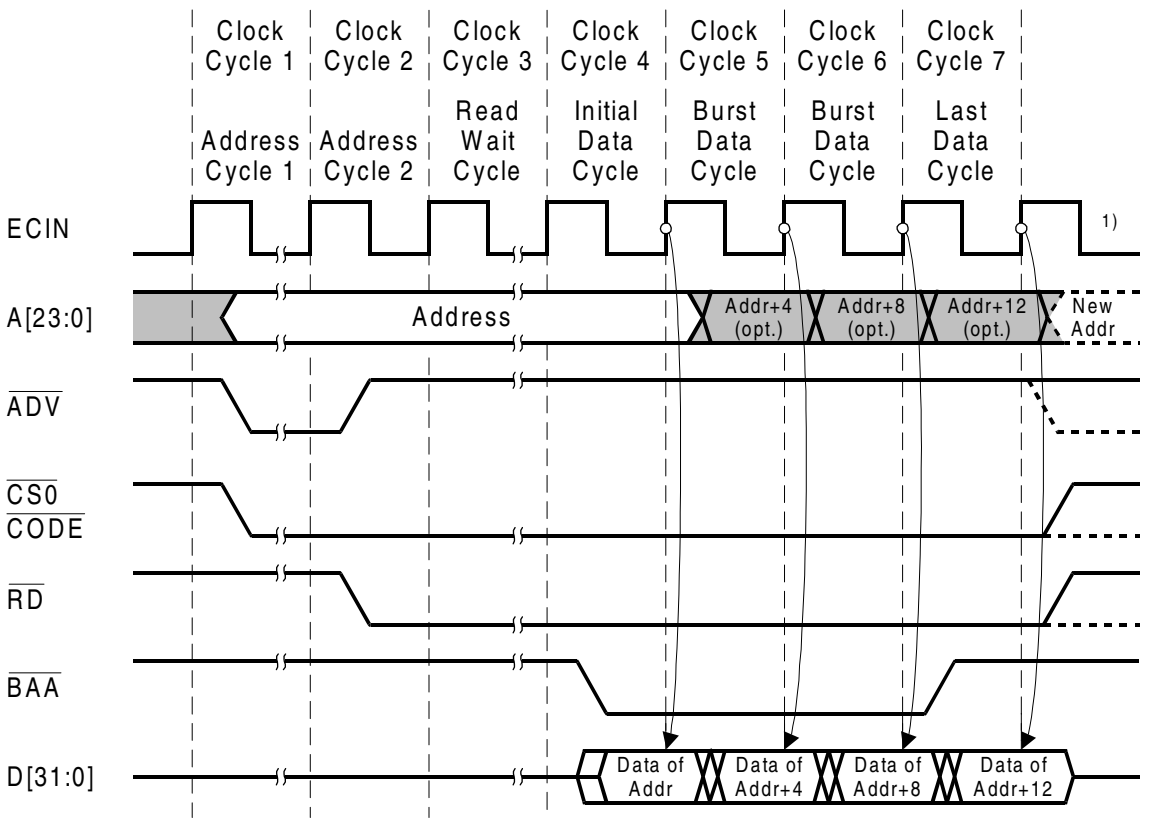
13.9.3 Cycle Definitions of Burst Mode Timing

See also the description of **Section 8.4** in this User's Manual.

The following definitions are used to describe the timings:

- Address cycle 1, programmable to either 1 or 2 (Clock Cycle 1 in **Figure 13-11**)
- Address cycle 2 (Clock Cycle 2 in **Figure 13-11**)
- Read wait cycle, programmable from 0-7 (Clock Cycle 3 in **Figure 13-11**)
- Initial data cycle, programmable to either 1 or 2 for one data access (Clock Cycle 4 in **Figure 13-11**)
- Burst data cycle, programmable to either 1 or 2 for one data access (Clock Cycle 5,6 in **Figure 13-11**)
- Last burst data cycle (Clock Cycle 7 in **Figure 13-11**)

The figures on the next pages show the basic timing of a synchronous burst mode operation.



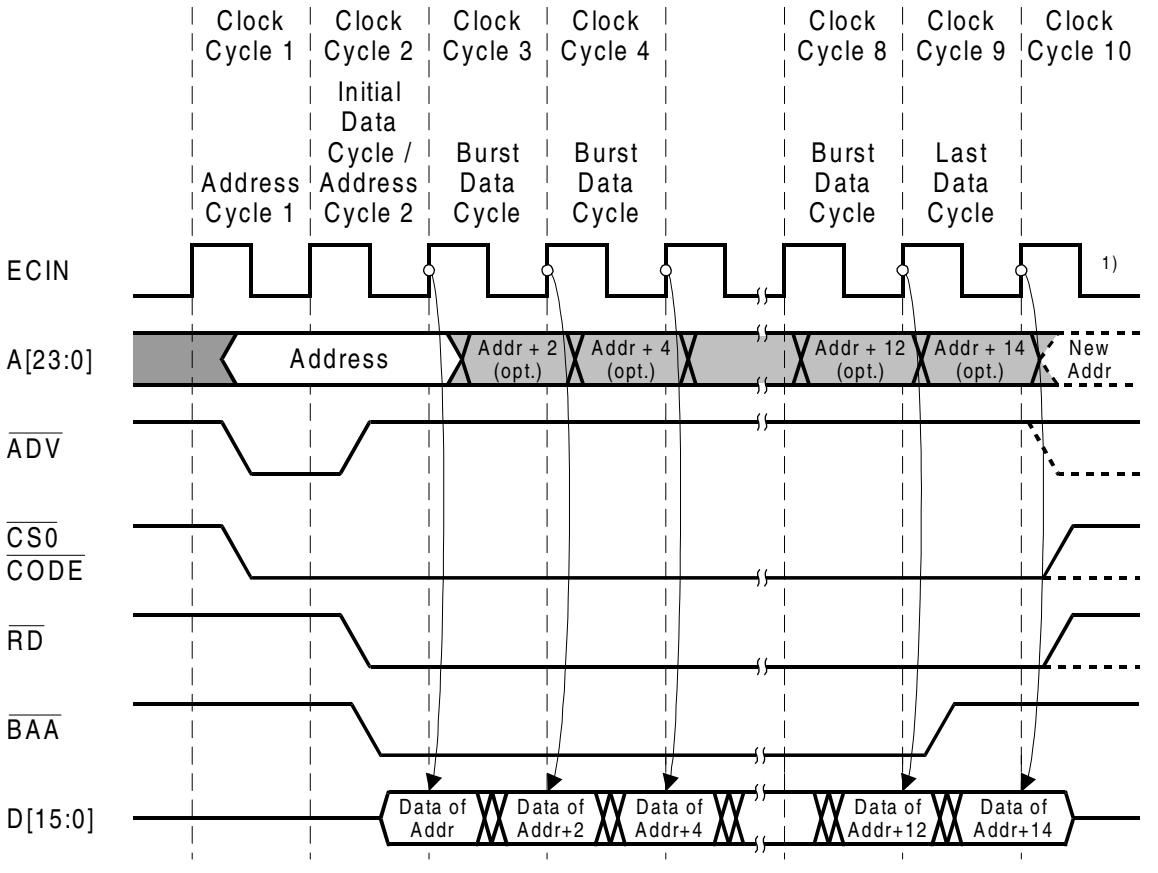
1) The dotted waveforms indicate the start of a new address cycle.

Example settings: Burst of 4 words, ADVLEN = 0, RDWLEN = 1, OIDA2 = 0
Burst data address output optional, enabled via bit DAEN.

MCT04999

Figure 13-11 Synchronous Burst Read Operation (4 Word Burst on 32-bit bus)

Address Cycle 1 (clock cycle 0) can be repeated once controlled by bit PMU_EIFCON.ADVLEN, Address Cycle 2 (clock cycle one in this picture) can be overlayed to the Initial Data Cycle by setting bit PMU_EIFCON.SIDC, if no Read Wait Cycles are programmed.



1) The dotted waveforms indicate the start of a new address cycle.

Example settings: Burst of 4 halfwords, ADVLEN = 0, RDWLEN = 0, OIDA2 = 0
Burst data address output optional, enabled via bit DAEN.

MCT04957

Figure 13-12 Synchronous Burst Read Operation (4 Word Burst on 16-bit Bus)

The delay between the address latch phase and the first data burst cycle (number of read wait cycles, cycle 2) can be programmed by bit field PMU_EIFCON.RDWLEN, This feature allows a flexible adoption of the initial read access time of a burst mode device. Further, the number of data cycles for one data access can be one or two ECOUT periods (defined by bit PMU_EIFCON.DATLEN).

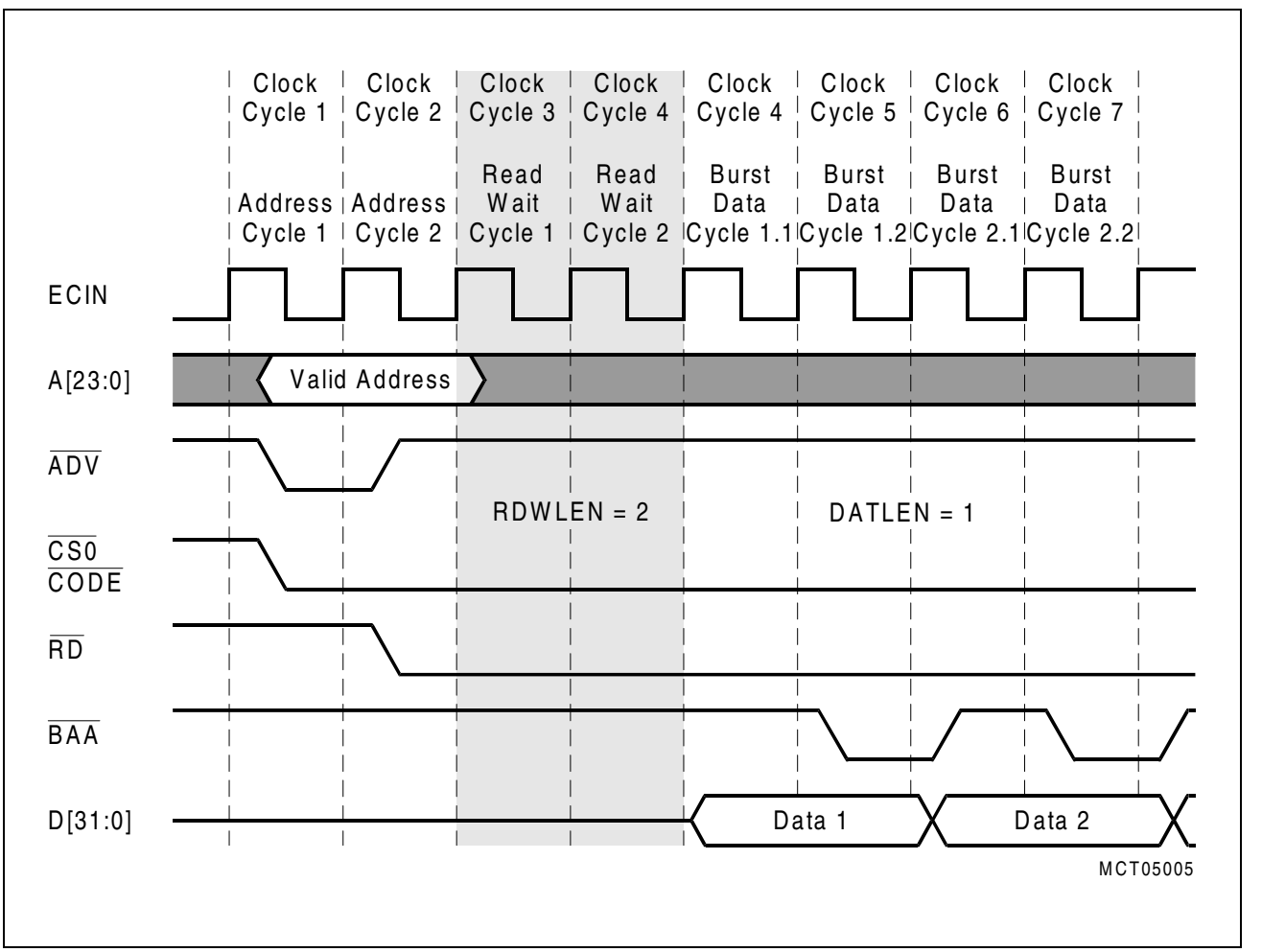


Figure 13-13 Programming of Read Wait Cycles and Data Cycles

Address Cycle one can be repeated once if the address setup time of the flash device requires this.

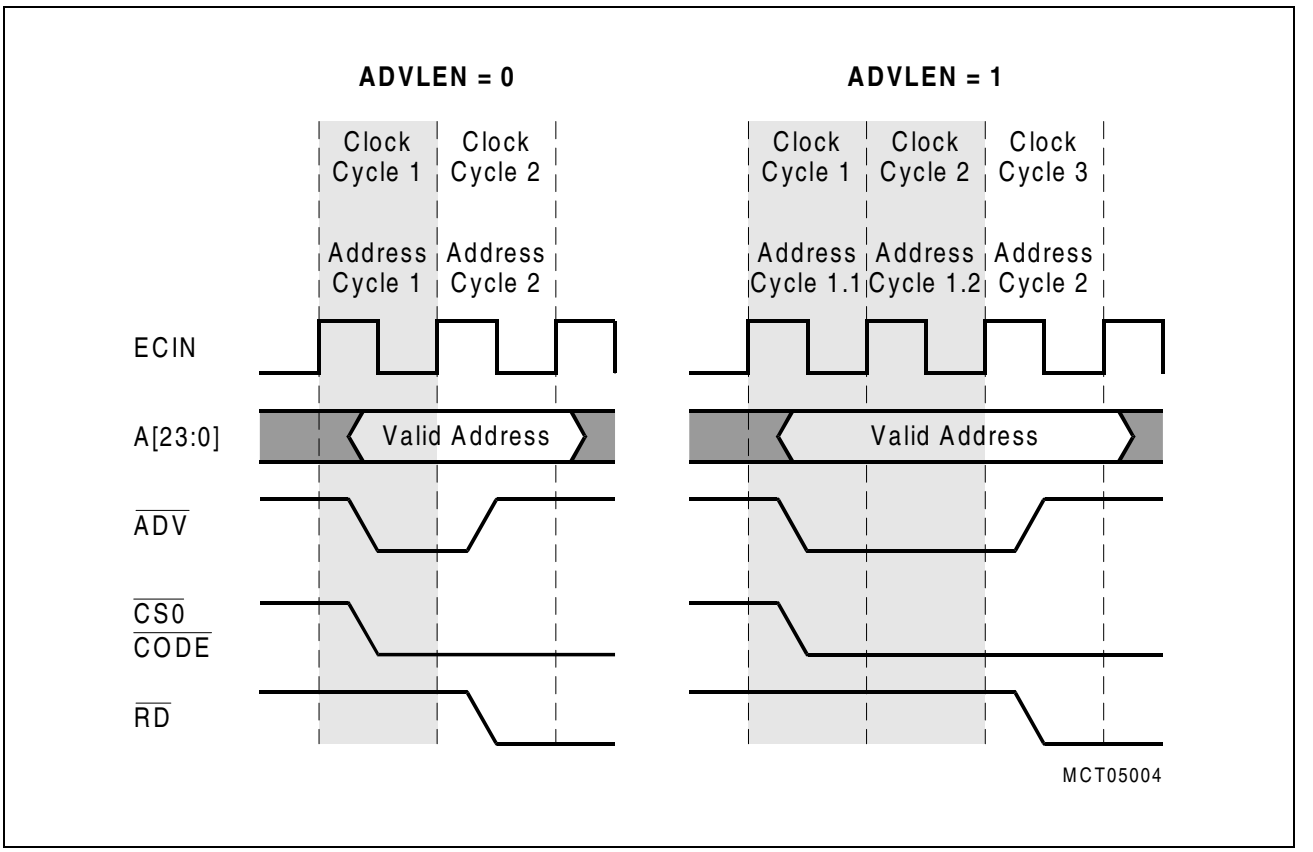


Figure 13-14 Programming of Address Cycle 1 Duration

For very fast devices like synchronous SRAMs the initial data cycle might be folded (overlayed) onto address cycle 2, resulting in very fast access times.

(If read wait cycles are programmed, setting of OIDA2 will result in reducing the wait time by one cycle).

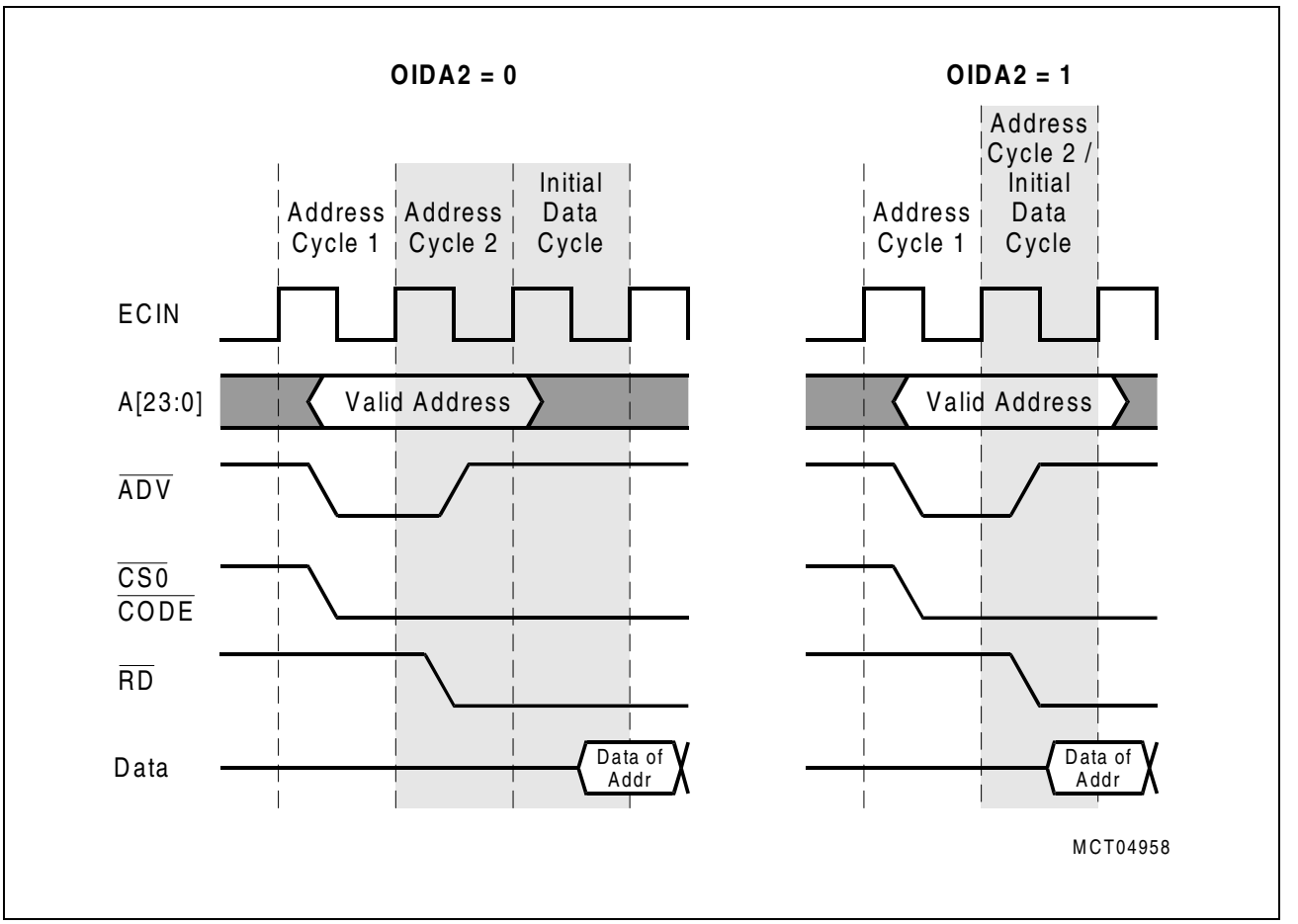


Figure 13-15 Overlaying Address Cycle 2 with Initial Data Cycle

If the $\overline{WAIT}/\overline{IND}$ input is defined for data wait cycle operation (PMU_EIFCON. WAITFUNC = 0), wait cycles can be inserted between burst data cycles. A wait cycle is inserted if a low level is detected at the end of a data cycle (Cycle 3 or 4). A high level at $\overline{WAIT}/\overline{IND}$ at the end of a data cycle proceeds the burst data cycles.

Figure 13-16 shows the burst mode wait cycle timing with DATLEN = 0. **Figure 13-17** shows the burst mode wait cycle timing with DATLEN = 1.

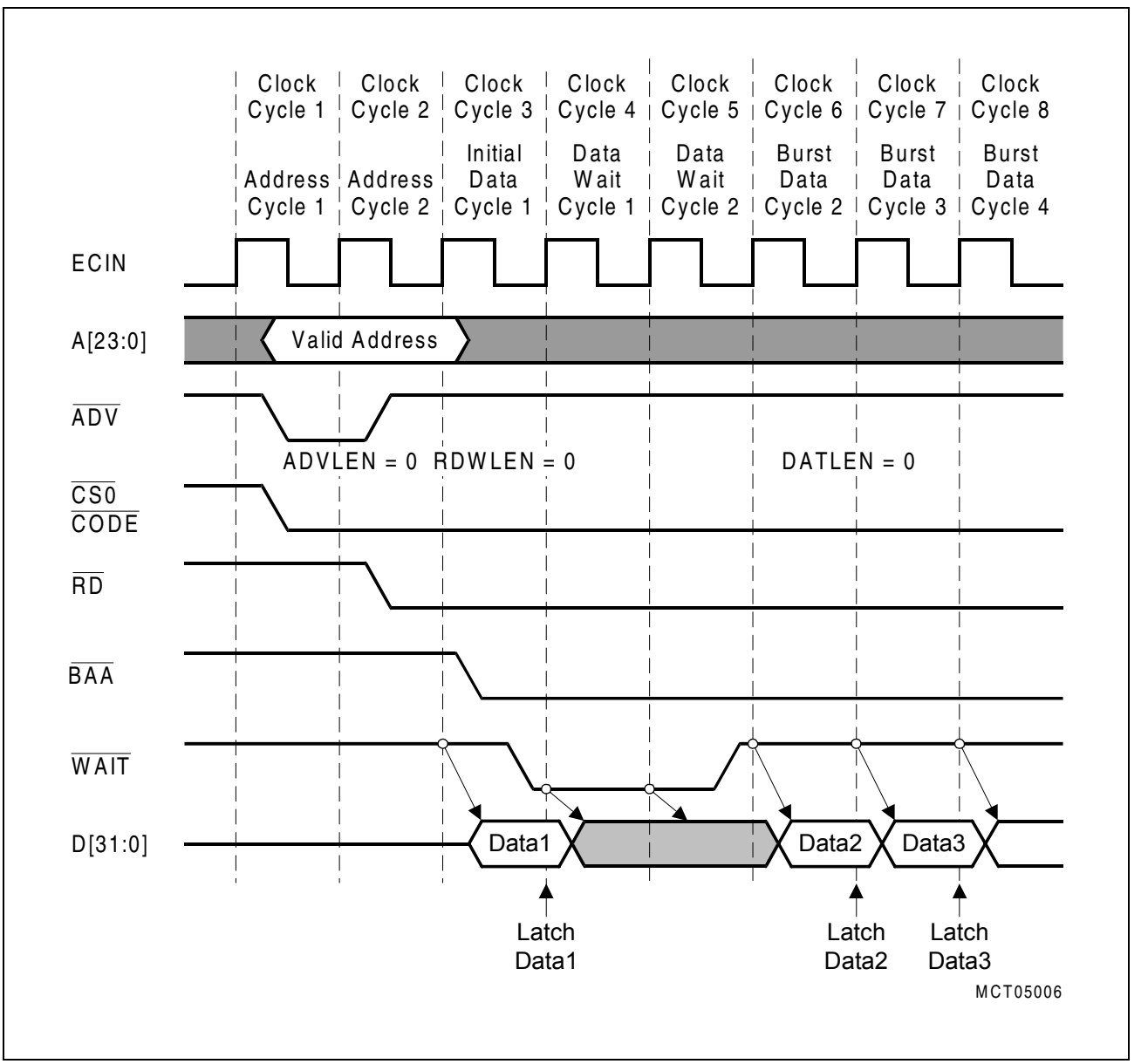


Figure 13-16 Inserting Wait Cycles within a Burst (PMU_EIFCON.WAITFUNC = 0 and PMU_EIFCON.DATLEN = 0)

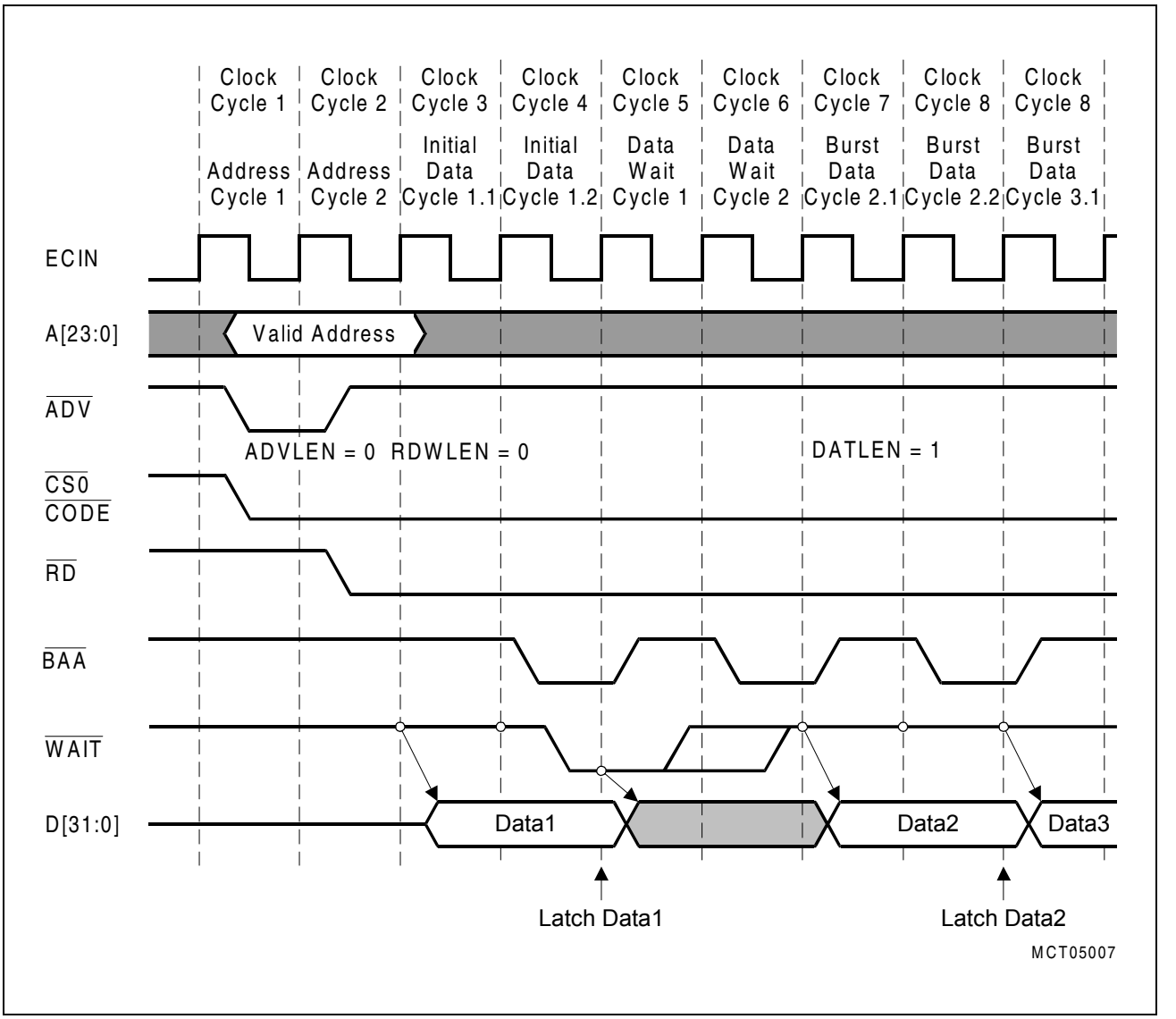


Figure 13-17 Inserting Wait Cycles within a Burst (PMU_EIFCON.WAITFUNC = 0 and PMU_EIFCON.DATLEN = 1)

If the $\overline{WAIT}/\overline{IND}$ input is assigned for terminate burst function (PMU_EIFCON. WAITFUNC = 1), consecutive data burst cycles are terminated if a low level is detected at the end of a data cycle (Cycle 3 or 4).

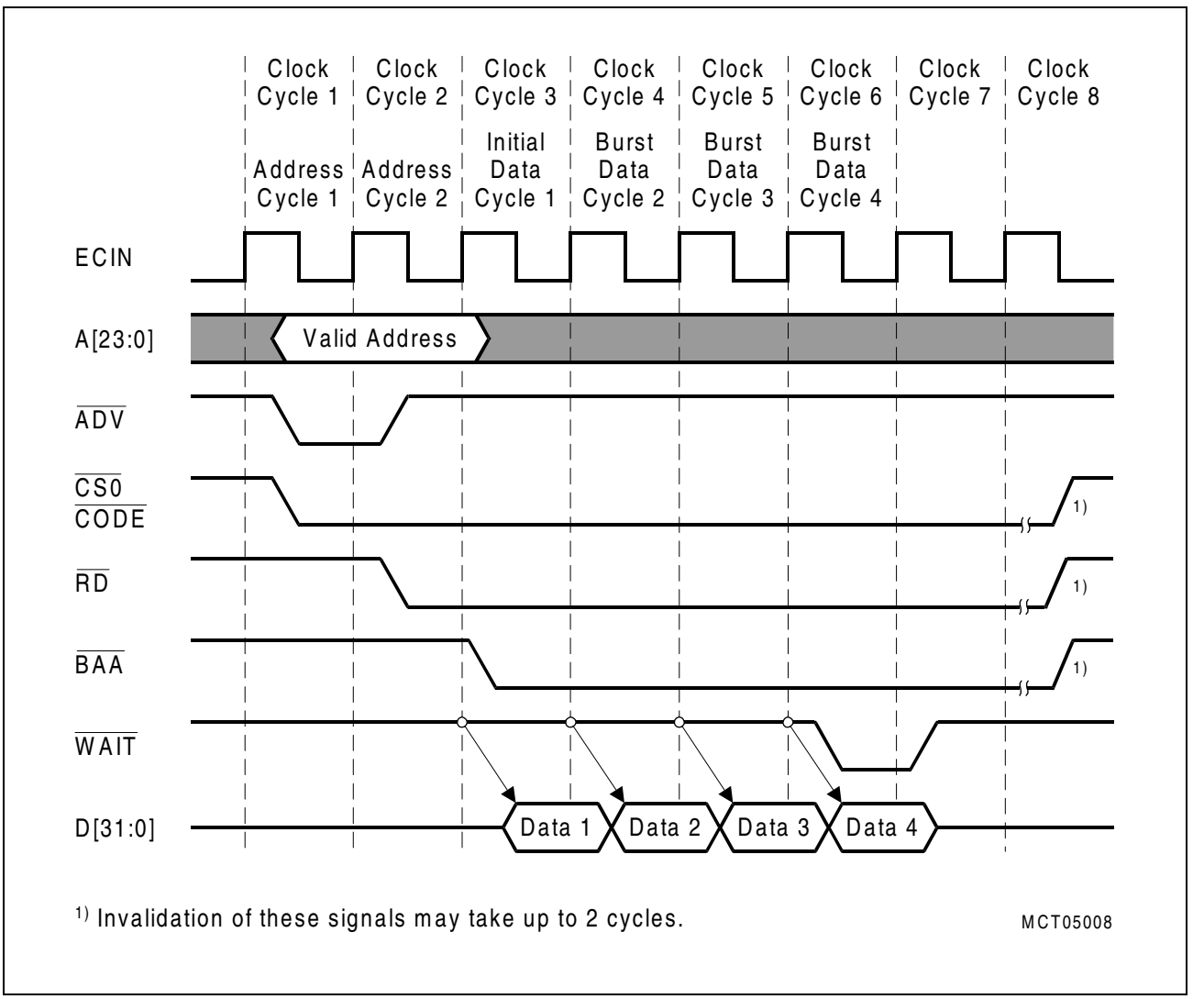


Figure 13-18 Terminating a Continuous Burst Operation with $\overline{WAIT}/\overline{IND}$ (PMU_EIFCON.WAITFUNC = 1)

13.10 Arbitration between EBU and PMU/BIFU for External Accesses

After reset with external boot selected, FPI Bus accesses have high priority until the boot process has been finished. Afterwards, burst mode accesses initiated by the PMU/BIFU have higher priority.

13.11 EBU Registers

This section describes the control registers and programmable parameters of the EBU. Figure 13-19 shows all FPI Bus accessible registers associated with the EBU. Register EBU_EXTCON, accessible from the external bus only, is described at the end.

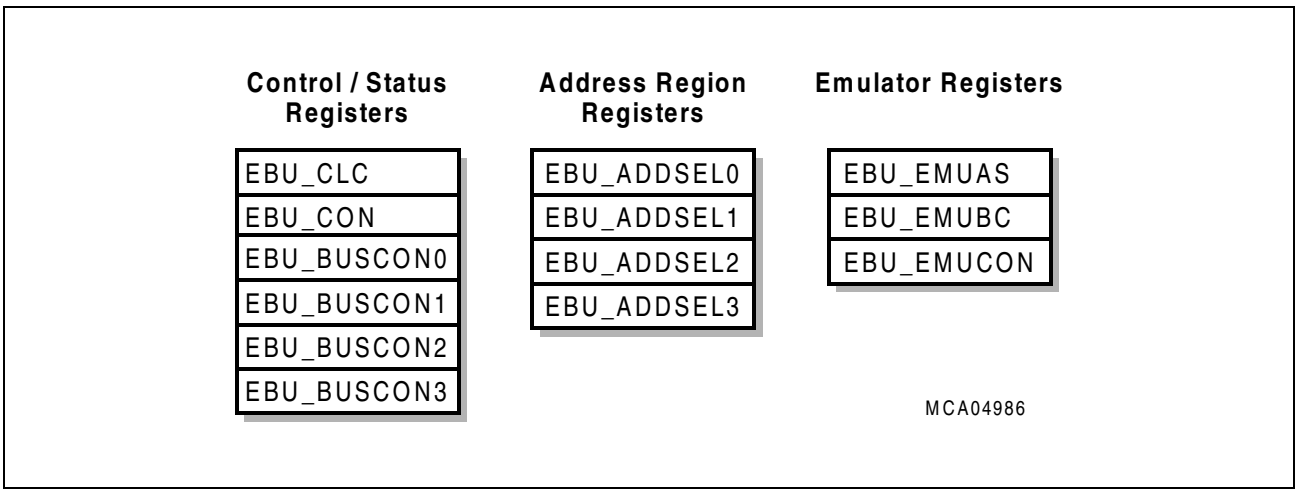


Figure 13-19 EBU Registers

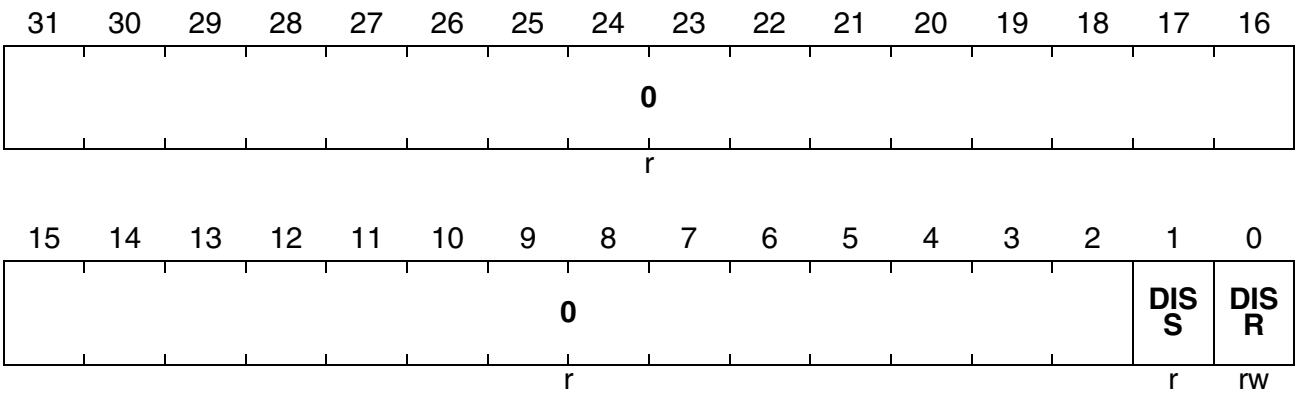
Table 13-10 EBU Registers

Register Short Name	Register Long Name	Offset Address	Description see
EBU_CLC	EBU Clock Control Register	0000_H	Page 13-44
EBU_CON	EBU Global Control Register	0010_H	Page 13-45
EBU_ADDSEL0	EBU Address Select Register 0	0020_H	Page 13-46
EBU_ADDSEL1	EBU Address Select Register 1	0024_H	
EBU_ADDSEL2	EBU Address Select Register 2	0028_H	
EBU_ADDSEL3	EBU Address Select Register 3	$002C_H$	
EBU_BUSCON0	EBU Bus Configuration Register 0	0060_H	Page 13-47
EBU_BUSCON1	EBU Bus Configuration Register 1	0064_H	
EBU_BUSCON2	EBU Bus Configuration Register 2	0068_H	
EBU_BUSCON3	EBU Bus Configuration Register 3	$006C_H$	
EBU_EMUAS	EBU Emulator Address Select Register	0080_H	Page 13-54
EBU_EMUBC	EBU Emulator Bus Configuration Register	0084_H	Page 13-51
EBU_EMUCON	EBU Emulator Configuration Register	0088_H	Page 13-50

13.11.1 Clock Control Register

The EBU clock control register EBU_CLC allows to enable/disable the EBU in general. After reset the EBU is enabled.

EBU_CLC
EBU Clock Control Register **Reset Value: 0000 0000$_H$**



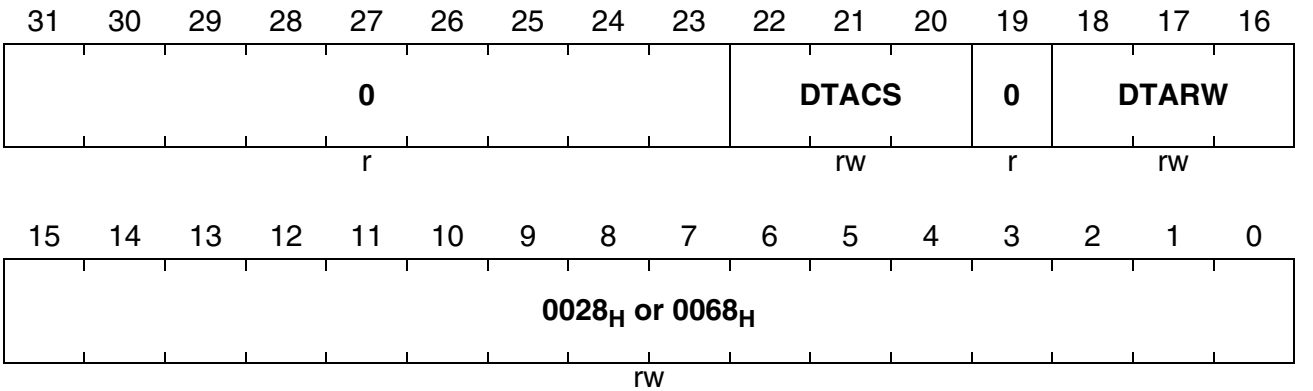
Field	Bits	Type	Description
DISR	0	rw	**EBU Disable Request Bit** Used for enable/disable control of the EBU. 0 EBU disable is not requested 1 EBU disable is requested
DISS	1	r	**EBU Disable Status Bit** Bit indicates the current status of the EBU. 0 EBU is enabled (default after reset) 1 EBU is disabled
0	[31:2]	r	**Reserved**; read as 0; should be written with 0.

13.11.2 Global Control Register

The EBU Global Control Register EBU_CON provides global control of the EBU.

EBU_CON
EBU Global Control Register

Reset Value (internal boot): 0000 0028$_H$
Reset Value (external boot): 0000 0068$_H$

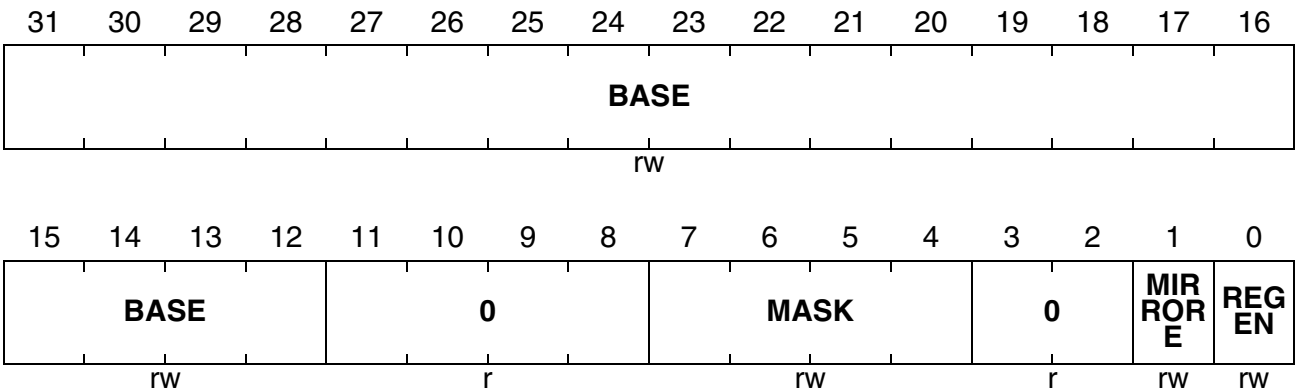


Field	Bits	Type	Description
–	[15:0]	rw	**Reserved**; read as 0028$_H$ or 0068$_H$; this bit field should be written with the value which is read.
DTARW	[18:16]	rw	**Driver Turn-Around Control, read-write triggered** Specifies the minimum number of inactive cycles between external accesses to the same device when switching from read to write or vice versa. n Insert n inactive cycles, where n is in the range of 0 - 7 (000$_B$ = default after reset)
DTACS	[22:20]	rw	**Driver Turn-Around Control, chip select triggered** Specifies the minimum number of inactive cycles between external accesses when switching from one chip select to another. n Insert n inactive cycles, where n is in the range of 0 - 7 (000$_B$ = default after reset)
0	19, [31:23]	r	**Reserved**; read as 0; should be written with 0.

13.11.3 Address Select Registers

The EBU Address Select Registers EBU_ADDSELx (x = 0-3) establish and control memory regions for external accesses.

EBU_ADDSELx (x = 3-0)
EBU Address Select Register x **Reset Value (internal boot): 0000 0000$_H$**
Reset Value (external boot): A000 0001$_H$



Field	Bits	Type	Description
REGEN	0	rw	**Memory Region Enable Control** 0 Memory region disabled 1 Memory region enabled
MIRRORE	1	rw	**Memory Region Mirror Enable Control** 0 Memory region is not mirrored into memory Segment 11 1 Memory region is mirrored into memory Segment 11
MASK	[7:4]	rw	**Memory Region Address Mask** Specifies the number of right-most bits in the base address that should be included in the address comparison, normally starting at bit position 26. Bits 31:27 should always be part of the address comparison.
BASE	[31:12]	rw	**Memory Region Base Address** FPI Bus addresses are compared to this base address in conjunction with the mask control.
0	2, 3, [11:8]	r	**Reserved**; read as 0; should be written with 0.

13.11.4 Bus Configuration Registers

The EBU Bus Configuration Registers EBU_BUSCONx (x = 3-0) configure access modes and access timing to the external memory regions defined through the EBU_ADDSELx registers.

Note: In case of an external boot, the reset value of EBU_BUSCON0 will be modified by the external boot configuration information retrieved from the External Boot Memory Configuration Word.

EBU_BUSCONx (x = 3-0)
EBU Bus Configuration Register x **Reset Value: E802 61FF$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WR DIS	1		BCGEN		ES	0		CMULTR		WAIT		WAI TINV	SET UP	PORTW	
rw	rw		rw		rw	rw		rw		rw		rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WAITRDC							WAITWRC			HOLDC		RECOVC		CMULT	
rw							rw			rw		rw		rw	

Field	Bits	Type	Description
CMULT	[1:0]	rw	**Wait Cycle Multiplier Control** Specifies a multiplier for the cycles specified via the WAITWRC, HOLDC and RECOVC fields. 00_B Multiplier is 1 01_B Multiplier is 4 10_B Multiplier is 8 11_B Multiplier is 16 (default after reset)
RECOVC	[3:2]	rw	**Recovery Cycle Control** Specifies the number of inactive cycles to be inserted after access to an external device. n Insert n inactive cycles after access, where n is in the range 0 - 3. (Default after reset is 3 cycles)
HOLDC	[5:4]	rw	**Hold/Pause Cycle Control** Controls number of hold cycles in DEMUXED mode. n Insert n cycles after access, where n is in the range 0 - 3. (Default after reset is 3 hold cycles.)

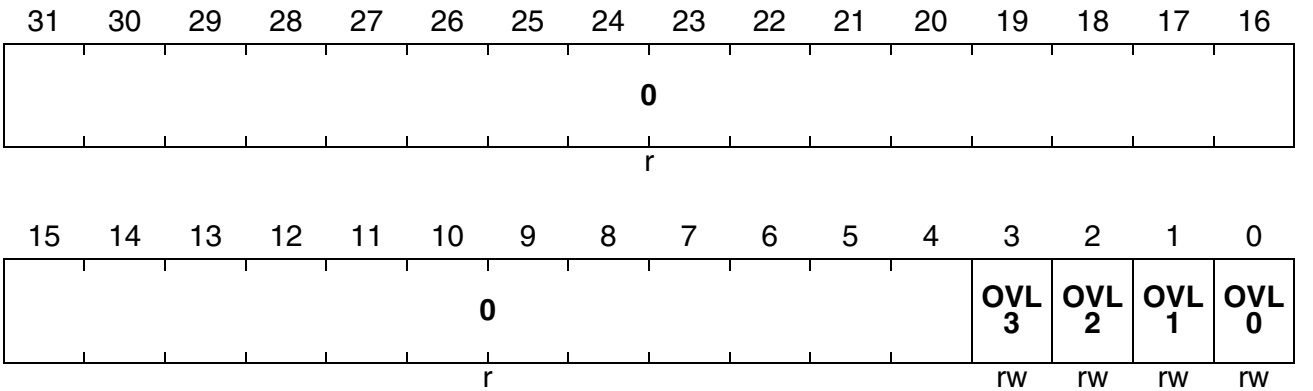
Field	Bits	Type	Description
WAITWRC	[8:6]	rw	**Write Access Wait-State Control** n Insert n wait states into write access, where n is in the range 0 - 7. (Default after reset is 7 wait states.)
WAITRDC	[15:9]	rw	**Read Access Wait-State Control** n Insert n wait states into read access, where n is in the range 0 - 127. (Default after reset is 48 wait states.)
PORTW x = 0	[17:16]	rw	**External Device Data Width Control (x = 0)** If SCU_CON.EXTIF = 0 (instruction fetches of the PMU via FPI Bus): 00_B 8-bit data 01_B 16-bit data 10_B 32-bit data (default after reset) 11_B Undefined, reserved If SCU_CON.EXTIF = 1 (burst instruction fetches of the PMU via BIFU): $0X_B$ 16-bit data $1X_B$ 32-bit data
PORTW x = 1, 2, 3			**External Device Data Width Control (x = 1, 2, 3)** 00_B 8-bit data 01_B 16-bit data 10_B 32-bit data (default after reset) 11_B Undefined, reserved
SETUP	18	rw	**Extended Address Setup Control** 0 Cycle 0 not generated (default after reset) 1 Cycle 0 generated
WAITINV	19	rw	**Active $\overline{\text{WAIT}}/\overline{\text{IND}}$ Level Control** 0 $\overline{\text{WAIT}}/\overline{\text{IND}}$ active low (default after reset) 1 $\overline{\text{WAIT}}/\overline{\text{IND}}$ active high
WAIT	[21:20]	rw	**Variable Wait-State Insertion Control** 00_B Variable wait-state insertion disabled (default after reset) 01_B Asynchronous wait-state insertion 10_B Synchronous wait-state insertion 11_B Undefined, reserved

Field	Bits	Type	Description
CMULTR	[23:22]	rw	**Cycle Multiplier Control for Read Cycles** Specifies a multiplier for the cycles specified by WAITRDC. 00_B Multiplier is 1 (default after reset) 01_B Multiplier is 2 10_B Multiplier is 4 11_B Multiplier is 8
0	[25:24]	rw	**Reserved**; read as 0 (default after reset); bits must be always written with 0.
ES	26	rw	**Early Read Sample Select** 0 Reads are performed with regular timing. Data is sampled with the falling edge of CLKOUT. (default after reset) 1 Early read sampling of data is enabled. Data is sampled with the rising edge of CLKIN.
BCGEN	[28:27]	rw	**Byte Control Signal Timing Mode Control** 00_B Chip Select Mode 01_B Control Mode (default after reset) 10_B Write Enable Mode 11_B Reserved
WRDIS	31	rw	**Memory Region Write Protection** 0 Writes to the memory region are enabled 1 Writes to the memory region are disabled (default after reset)
1	[30:29]	r	**Reserved**; bits with no function; writing to these bits stores the value which is written; default after reset is 1.

13.11.5 Emulator Configuration Register

The EBU Emulator Configuration Register EBU_EMUCON provides overlay memory control to the emulator.

EBU_EMUCON
EBU Emulator Configuration Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
OVL0	0	rw	**Overlay Memory Control for Region 0** 0 Do not activate $\overline{\text{CSEMU}}$/$\overline{\text{CSOVL}}$ on an access to region 0 1 Activate $\overline{\text{CSOVL}}$ on an access to region 0
OVL1	1	rw	**Overlay Memory Control for Region 1** 0 Do not activate $\overline{\text{CSOVL}}$ on an access to region 1 1 Activate $\overline{\text{CSOVL}}$ on an access to region 1
OVL2	2	rw	**Overlay Memory Control for Region 2** 0 Do not activate $\overline{\text{CSOVL}}$ on an access to region 2 1 Activate $\overline{\text{CSOVL}}$ on an access to region 2
OVL3	3	rw	**Overlay Memory Control for Region 3** 0 Do not activate $\overline{\text{CSOVL}}$ on an access to region 3 1 Activate $\overline{\text{CSOVL}}$ on an access to region 3
0	[31:4]	r	**Reserved**; read as 0; should be written with 0.

13.11.6 Emulator Bus Configuration Register

The EBU Emulator Bus Configuration Register EBU_EMUBC defines the access parameters for the emulator memory region determined through register EBU_EMUAS. This register has the same layout and semantics as the EBU_BUSCONx registers.

EBU_EMUBC
EBU Emulator Bus Configuration Register **Reset Value: 0016 0280$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WR DIS	0		BCGEN		ES	0		CMULTR		WAIT		WAI TINV	SET UP	PORTW	
rw	rw		rw		rw	rw		rw		rw		rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WAITRDC							WAITWRC			HOLDC		RECOVC		CMULT	
rw							rw			rw		rw		rw	

Field	Bits	Type	Description
CMULT	[1:0]	rw	**Wait Cycle Multiplier Control** Specifies a multiplier for the cycles specified via the WAITWRC, HOLDC and RECOVC fields. 00_B Multiplier is 1 (default after reset) 01_B Multiplier is 4 10_B Multiplier is 8 11_B Multiplier is 16
RECOVC	[3:2]	rw	**Recovery Cycle Control** Specifies the number of inactive cycles to be inserted after access to an external device. n Insert n inactive cycles after access, where n is in the range 0 - 3. (Default after reset is 0 cycles)
HOLDC	[5:4]	rw	**Hold/Pause Cycle Control** Controls number of hold cycles in DEMUXED mode. n Insert n cycles after access, where n is in the range 0 - 3. (Default after reset is 0 hold cycles.)
WAITWRC	[8:6]	rw	**Write Access Wait-State Control** n Insert n wait states into write access, where n is in the range 0 - 7. (Default after reset is 2 wait states.)

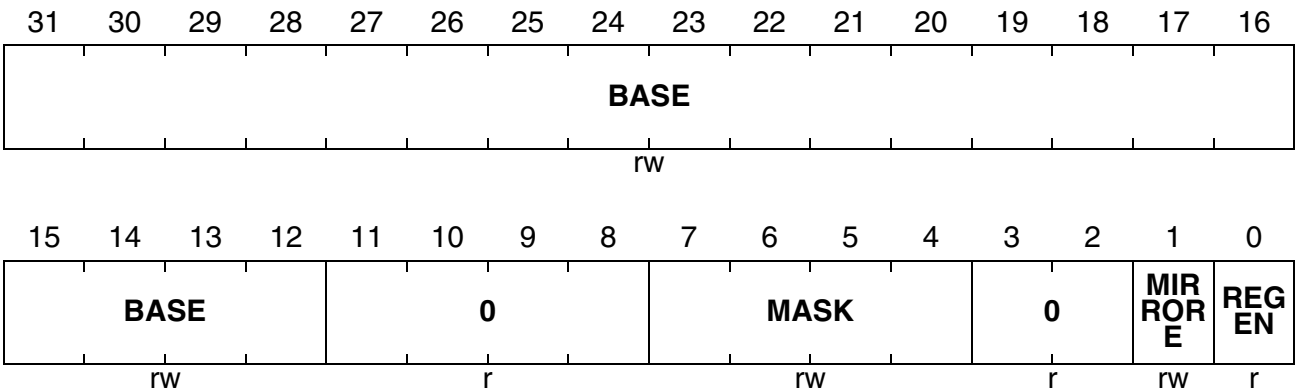
Field	Bits	Type	Description
WAITRDC	[15:9]	rw	**Read Access Wait-State Control** n Insert n wait states into read access, where n is in the range 0 - 127. (Default after reset is 1 wait state)
PORTW	[17:16]	rw	**External Device Data Width Control** 00_B 8-bit data 01_B 16-bit data 10_B 32-bit data (default after reset) 11_B Undefined, reserved
SETUP	18	rw	**Extended Address Setup Control** 0 Cycle 0 not generated 1 Cycle 0 generated (default after reset)
WAITINV	19	rw	**Active $\overline{\text{WAIT}}/\overline{\text{IND}}$ Level Control** 0 $\overline{\text{WAIT}}/\overline{\text{IND}}$ active low (default after reset) 1 $\overline{\text{WAIT}}/\overline{\text{IND}}$ active high
WAIT	[21:20]	rw	**Variable Wait-State Insertion Control** 00_B Variable wait-state insertion disabled 01_B Asynchronous wait-state insertion (default after reset) 10_B Synchronous wait-state insertion 11_B Undefined, reserved
CMULTR	[23:22]	rw	**Cycle Multiplier Control for Read Cycles** Specifies a multiplier for the cycles specified by WAITRDC. 00_B Multiplier is 1 (default after reset) 01_B Multiplier is 2 10_B Multiplier is 4 11_B Multiplier is 8
0	[25:24]	rw	**Reserved**; read as 0 (default after reset); bits must be always written with 0.
ES	26	rw	**Early Read Sample Select** 0 Reads are performed with regular timing (default after reset) 1 Early read sampling of data is enabled

Field	Bits	Type	Description
BCGEN	[28:27]	rw	**Byte Control Signal Timing Mode Control** 00_B Chip Select Mode (default after reset) 01_B Control Mode 10_B Write Enable Mode 11_B Reserved
ALEC	[30:29]	rw	**Address Latch Enable (ALE) Duration Control** n Insert *n* additional address setup cycles for accesses to a multiplexed device, where *n* is in the range 0 - 3. (Default after reset is 0 cycles.)
WRDIS	31	rw	**Memory Region Write Protection** 0 Writes to the memory region are enabled (default after reset) 1 Writes to the memory region are disabled
0	[30:29]	r	**Reserved**; bits with no function; writing to these bits stores the value which is written; default after reset is 0.

13.11.7 Emulator Address Select Register

The EBU Emulator Address Select Register EBU_EMUAS defines the address region for the emulator memory. This register has the same layout and semantics as the EBU_ADDSELx registers.

EBU_EMUAS
EBU Emulator Address Select Register **Reset Value: BE00 0031$_H$**



Field	Bits	Type	Description
REGEN	0	r	**Emulator Memory Region Enable Control** 0 Memory region disabled 1 Memory region enabled (default after reset)
MIRRORE	1	rw	**Emulator Memory Region Mirror Enable Control** 0 Memory region is not mirrored into memory Segment 11 (default after reset) 1 Memory region is mirrored into memory Segment 11
MASK	[7:4]	rw	**Emulator Memory Region Address Mask** Specifies the number of right-most bits in the base address starting at bit position 26, which should be included in the address comparison. Bits [31:27] are always part of the address comparison. (default after reset = 0011_B)
BASE	[31:12]	rw	**Emulator Memory Region Base Address** Any FPI address is compared to this base address in conjunction with mask control.
0	2, 3, [11:8]	r	**Reserved**; read as 0; should be written with 0.

13.11.8 EBU Register Address Range

In the TC1765, the registers of the EBU are located in the following address range:

– Module Base Address. F000 0500_H
  Module End Address. F000 $05FF_H$
– Absolute Register Address = Module Base Address + Offset Address
  (offset addresses see **Table 13-10**)

14 Interrupt System

The TC1765 interrupt system provides a flexible and time-efficient means for processing interrupts. This chapter describes the interrupt system for the TC1765. Topics covered include the architecture of the interrupt system, interrupt system configuration, and the interrupt operations of the TC1765 peripherals and Central Processing Unit (CPU).

14.1 Overview

An interrupt request is serviced by the CPU. Interrupt requests are also called “Service Requests” because they are serviced by a “Service Provider”, the CPU.

Each peripheral in the TC1765 can generate service requests and. Additionally, the Bus Control Unit, the Debug Unit, the DMA controller, and even the CPU itself can generate service requests. Several peripheral units are able to generate in parallel to a service request DMA requests to the DMA Controller. See **Chapter 10** for more details.

As shown in **Figure 14-1**, each TC1765 unit that can generate service requests is connected to one or multiple Service Request Nodes (SRN). Each SRN contains a Service Request Control Register mod_SRCx, where “mod” is the identifier of the service requesting unit and “x” an optional index. The Interrupt arbitration bus connects the SRNs with the Interrupt Control Unit, which handles interrupt arbitration among competing interrupt service requests.

Units which can generate service requests are:

- General Purpose Timer Unit (GPTU) with 8 SRNs
- General Purpose Timer Array (GPTA) with 54 SRNs
- Two High-Speed Synchronous Serial Interfaces (SSC0/SSC1) with 3 SRNs each
- Two Asynchronous/Synchronous Serial Interfaces (ASC0/ASC1) with 4 SRNs each
- TwinCAN controller with 8 SRNs
- Two Analog/Digital Converters (ADC0/ADC1) with 4 SRNs each
- Bus Control Unit (BCU) with 1 SRN
- DMA Controller Processor (DMA) with 8 SRNs
- Central Processing Unit (CPU) with 4 SRNs
- Debug Unit (OCDS) with 1 SRN
- Central Processing Unit (CPU) with 4 SRNs (software activated)

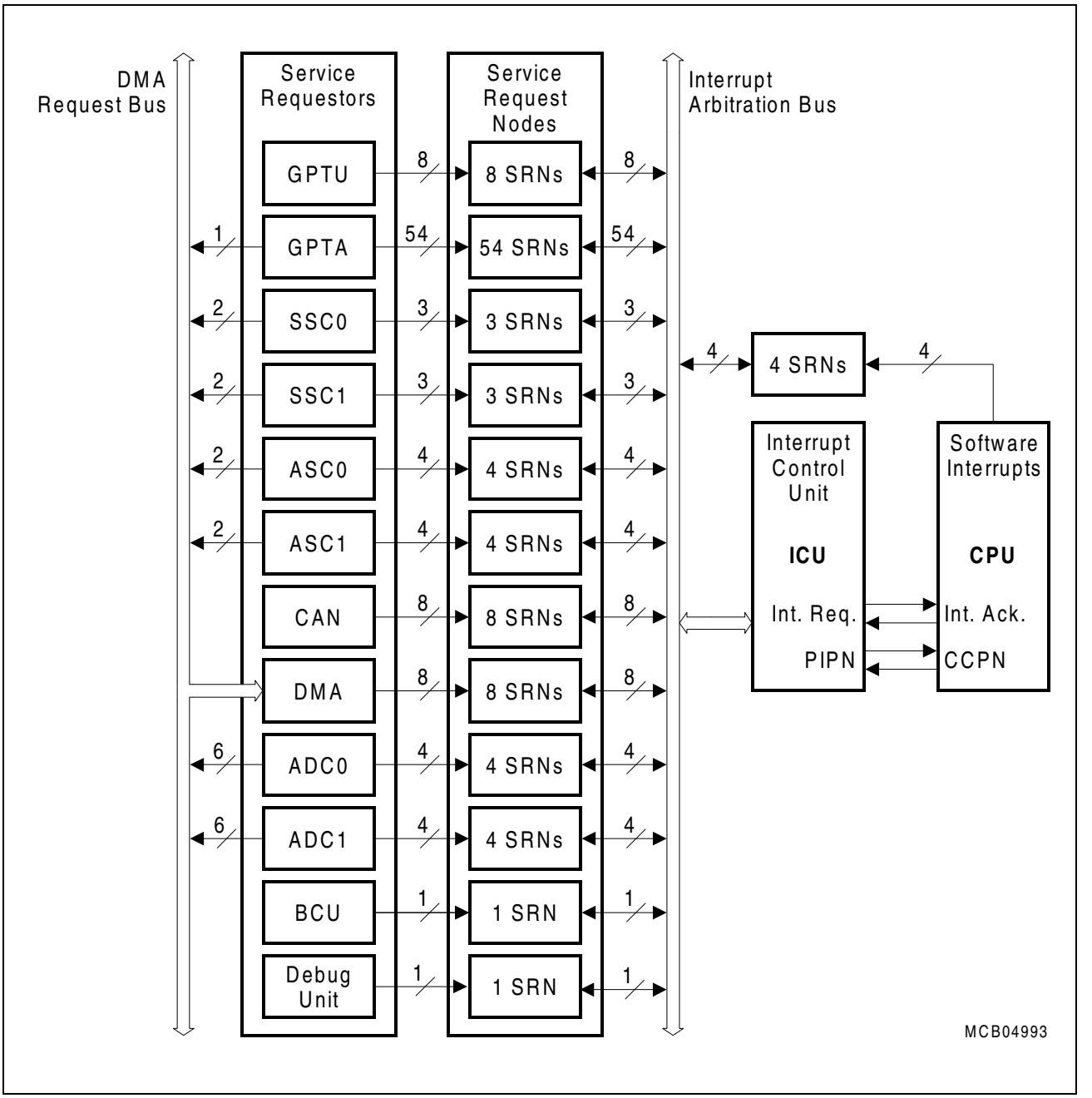


Figure 14-1 Block Diagram of the TC1765 Interrupt System

14.2 External Interrupts

External interrupt inputs in TC1765 are available using the input pins connected to the General Purpose Timer Unit (GPTU). Each of the seven GPTU I/O pins can be used as an external interrupt input, using the Service Request Nodes of the GPTU module. Additionally, such an external interrupt input can also trigger a timer function.

14.3 Service Request Nodes

In total, there are 110 Service Request Nodes available in the TC1765. Each Service Request Node contains a Service Request Control Register. Some peripheral units of the TC1765 have multiple Service Request Nodes. Note that the four CPU Service Request Nodes can be activated only by software.

14.3.1 Service Request Control Registers

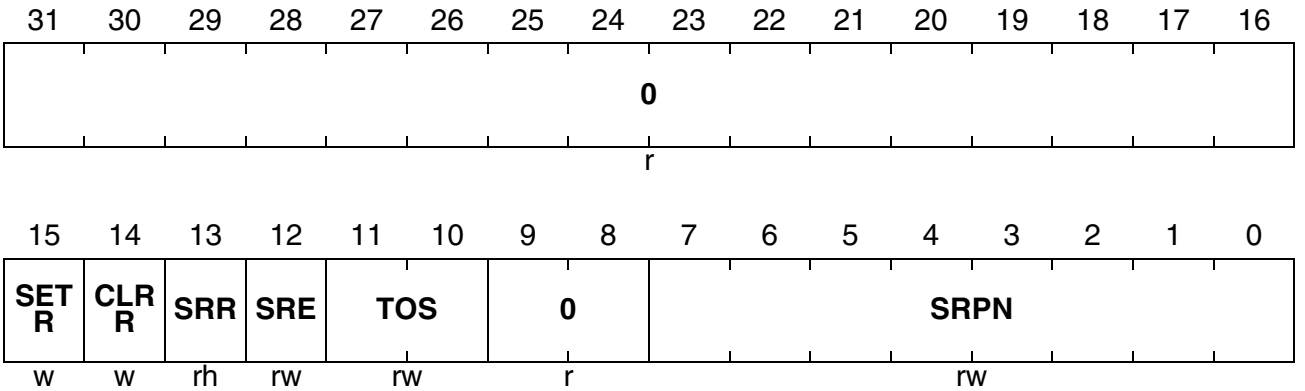
All Service Request Control Registers in the TC1765 have the same format. In general, these registers contain:

- Enable/disable information
- Priority information
- Service Provider destination
- Service request active status bit
- Software-initiated service request set and reset bits

Besides being activated by the associated triggering unit through hardware, each SRN can also be set or reset by software via two software-initiated service request control bits.

Note: The description given in this chapter characterizes all Service Request Control Registers of the TC1765. Informations on further peripheral module interrupt functions, such as enable or request flags, are described in the corresponding chapters of the peripheral modules.

mod_SRC
Service Request Control Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number** 00_H Service request is never serviced 01_H Service request is on lowest priority … … FF_H Service request is on highest priority
TOS	[11:10]	rw	**Type of Service Control** 00_B CPU service is initiated Other bit combinations of TOS must not be used.
SRE	12	rw	**Service Request Enable** 0 Service request is disabled 1 Service request is enabled
SRR	13	rh	**Service Request Flag** 0 No service request is pending 1 A service request is pending
CLRR	14	w	**Request Clear Bit** CLRR is required to reset SRR. 0 No action 1 Clear SRR; bit value is not stored; read always returns 0; no action if SETR is set also.
SETR	15	w	**Request Set Bit** SETR is required to set SRR. 0 No action 1 Set SRR; bit value is not stored; read always returns 0; no action if CLRR is set also.
0	[9:8], [31:16]	r	**Reserved**; read as 0; should be written with 0.

14.3.1.1 Service Request Flag (SRR)

A trigger event in a peripheral associated with this register causes SRR to be set to 1. Service requests can be acknowledged automatically by hardware or can be polled by software. If the corresponding enable bit SRE is set, a service request will be forwarded for arbitration to the Interrupt Control Unit (ICU). When the service request is acknowledged this bit is reset by hardware to 0.

The SRR bit can also be monitored, set, and reset by software via the SETR or CLRR bits respectively. This allows software to poll for events in peripheral devices. Writing directly to SRR via software has no effect.

14.3.1.2 Request Set and Clear Bits (SETR, CLRR)

The SETR and CLRR bits allow software to set or clear the service request bit SRR. Writing a 1 to SETR causes bit SRR to be set to 1. Writing a 1 to CLRR causes bit SRR to be cleared to 0. If hardware attempts to modify SRR during an atomic read-modify-write software operation (such as the bit-set or bit-clear instructions) the software operation will succeed and the hardware operation will have no effect.

The value written to SETR or CLRR is not stored. Writing a 0 to these bits has no effect. These bits always return 0 when read. If both, SETR and CLRR are set to 1 at the same time, SRR is not changed.

14.3.1.3 Enable Bit (SRE)

The SRE bit enables an interrupt to take part in the arbitration. It does not enable or disable the setting of the request flag SRR; the request flag can be set by hardware or by software (via SETR) independent of the state of the SRE bit. This allows service requests to be handled automatically by hardware or through software polling.

If SRE = 1, pending service requests are directed to the ICU for interrupt arbitration. The SRR bit is automatically set to 0 by hardware when the service request is acknowledged and serviced. It is recommended that in this case, software should not modify the SRR bit to avoid unexpected behavior due to the hardware controlling this bit.

If SRE = 0, pending service requests are not directed to the ICU. Software can poll the SRR bit to check whether a service request is pending. To acknowledge the service request, the SRR bit must then be reset by software by writing a 1 to CLRR.

Note: In this document, ‘active source’ means a Service Request Node whose Service Request Control Register has its request enable bit SRE set to 1 to allow its service requests to participate in interrupt arbitration.

14.3.1.4 Service Request Flag (SRR)

When set, the SRR flag indicates that a service request is pending. It can be set or reset directly by hardware or indirectly through software using the SETR and CLRR bits. Writing directly to this bit via software has no effect.

The SRR status bit can be directly set or reset by the associated hardware. For instance, in the General Purpose Timer Unit, an associated timer event can cause this bit to be set to 1. The details of how hardware events can cause the SRR bit to be set are defined in the individual peripheral chapters.

The acknowledgment of the service request by the Interrupt Control Unit (ICU) causes the SRR bit to be cleared.

SRR can be set or cleared either by hardware or by software regardless of the state of the enable bit SRE. However, the request is only forwarded for service if the enable bit is set. If SRE = 1, a pending service request takes part in the interrupt arbitration of the service provider selected by the device's TOS field. If SRE = 0, a pending service request is excluded from interrupt arbitrations.

SRR is automatically reset by hardware when the service request is acknowledged and serviced. Software can poll SRR to check for a pending service request. SRR must be reset by software in this case by writing a 1 to CLRR.

14.3.1.5 Type-of-Service Control (TOS)

The Service Provider for service requests in the TC1765 is the CPU. The TOS bit field is used to select whether a service request generates an interrupt to the CPU (TOS = 00_B). In the TC1765, other bit combinations of TOS are reserved for extensions of the interrupt system and must not be used.

14.3.1.6 Service Request Priority Number (SRPN)

The 8-bit Service Request Priority Number (SRPN) indicates the priority of a interrupt request with respect to other sources requesting interrupt.

Each active source must have a unique SRPN value to differentiate its priority. The special SRPN value of 00_H excludes an SRN from taking part in arbitration, regardless of the state of its SRE bit. If a source is not active — meaning its SRE bit is 0 — no restrictions are applied to the service request priority number.

The SRPN is used by the ICU to select an Interrupt Service Routine (ISR) to service the request. ISRs are associated with Service Request Priority Numbers by an Interrupt Vector Table. This means that the TC1765 Interrupt Vector Table is ordered by priority number. This is unlike traditional interrupt architectures in which their interrupt vector tables are ordered by the source of the interrupt. The TC1765 Interrupt Vector Table allows a single peripheral can have multiple priorities for different purposes.

The range of values for SRPNs used in a system depends on the number of possible active service requests and the user-definable organization of the Interrupt Vector Table. The 8-bit SRPNs permit up to 255 sources to be active at one time (remembering that the special SRPN value of 00_H excludes an SRN from taking part in arbitration).

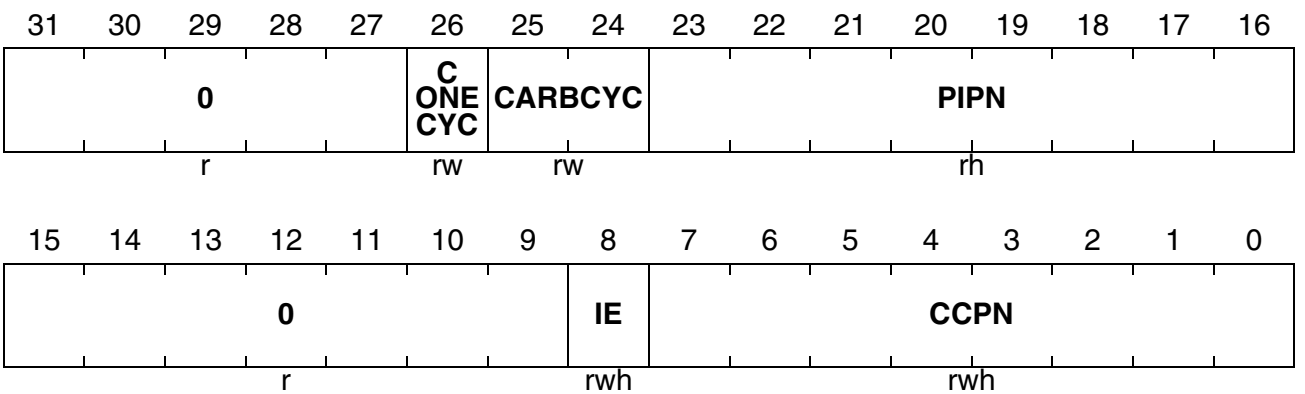
14.4 Interrupt Control Unit (ICU)

The Interrupt Control Unit manages the interrupt system, arbitrates incoming service requests, and determine whether and when the interrupt is directed to the CPU (see **Figure 14-1**).

14.4.1 ICU Interrupt Control Register (ICR)

The ICU Interrupt Control Register ICR holds the current CPU priority number (CCPN), the global interrupt enable/disable bit (IE), the pending interrupt priority number (PIPN), and bit fields which control the interrupt arbitration process.

ICR
ICU Interrupt Control Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CCPN	[7:0]	rwh	**Current CPU Priority Number** The Current CPU Priority Number (CCPN) bit field indicates the current priority level of the CPU. It is automatically updated by hardware on entry and exit of interrupt service routines, and through the execution of a BISR instruction. CCPN can also be updated through an MTCR instruction.

Field	Bits	Type	Description
IE	8	rwh	**Global Interrupt Enable Bit** The interrupt enable bit globally enables the CPU service request system. Whether a service request is delivered to the CPU depends on the individual Service Request Enable Bits (SRE) in the SRNs, and the current state of the CPU. IE is automatically updated by hardware on entry and exit of an Interrupt Service Routine (ISR). IE is cleared to 0 when an interrupt is taken, and is restored to the previous value when the ISR executes an RFE instruction to terminate itself. IE can also be updated through the execution of the ENABLE, DISABLE, MTCR, and BISR instructions. 0 Interrupt system is globally disabled 1 Interrupt system is globally enabled
PIPN	[23:16]	rh	**Pending Interrupt Priority Number** PIPN is a read-only bit field that is updated by the ICU at the end of each interrupt arbitration process. It indicates the priority number of the pending service request. PIPN is set to 0 when no request is pending, and at the beginning of each new arbitration process. 00_H No valid pending request YY_H A request with priority YY_H is pending
CARBCYC	[25:24]	rw	**Number of Arbitration Cycles** CARBCYC controls the number of arbitration cycles used to determine the request with the highest priority. 00_B 4 arbitration cycles (default) 01_B 3 arbitration cycles 10_B 2 arbitration cycles 11_B 1 arbitration cycle
CONECYC	26	rw	**Number of Clocks per Arbitration Cycle Control** The CONECYC bit determines the number of system clocks per arbitration cycle. This bit should be set to 1 only for system designs utilizing low system clock frequencies. 0 2 clocks per arbitration cycle (default) 1 1 clock per arbitration cycle
0	[15:9], [31:27]	r	**Reserved**; read as 0; should be written with 0.

14.4.1.1 Operation of the Interrupt Control Unit (ICU)

Service-request arbitration is performed in the ICU in parallel with normal CPU operation. When a triggering event occurs in one or more interrupt sources, the associated SRNs, if enabled, send service requests to the CPU through the ICU. The ICU determines which service request has the highest priority. The ICU will then forward the service request to the CPU. The service request will be acknowledged by the CPU and serviced, depending upon the state of the CPU.

The ICU arbitration process takes place in one or more arbitration cycles over the CPU Interrupt Arbitration Bus. The ICU begins a new arbitration process when a new service request is detected. At the end of the arbitration process, the ICU will have determined the service request with the highest priority number. This number is stored in the ICR.PIPN bit field and becomes the pending service request.

After the arbitration process, the ICU forwards the pending service request to the CPU by attempting to interrupt it. The CPU can be interrupted only if interrupts are enabled globally (that is, ICR.IE = 1) and if the priority of the service request is higher than the current processor priority (ICR.PIPN > ICR.CCPN). Also, the CPU may be temporarily blocked from taking interrupts, for example, if it is executing a multi-cycle instruction such as an atomic read-modify-write operation. The full list of conditions which could block the CPU from immediately responding to an interrupt request generated by the ICU is:

- Current CPU priority, ICR.CCPN, is equal to or higher than the pending interrupt priority, ICR.PIPN
- Interrupt system is globally disabled (ICR.IE = 0)
- CPU is in the process of entering an interrupt- or trap-service routine
- CPU is executing non-interruptible trap services
- CPU is executing a multi-cycle instruction
- CPU is executing an instruction which modifies the conditions of the global interrupt system, such as modifying the ICR
- CPU detects a trap condition (such as context depletion) when trying to enter a service routine

When the CPU is not otherwise prevented from taking an interrupt, the CPU's program counter will be directed to the Interrupt Service Routine entry point associated with the priority of the service request. Now, the CPU saves the value of ICR.PIPN internally, and acknowledges the ICU. The ICU then forwards the acknowledge signal back to the SRN that is requesting service, to inform it that it will be serviced by the CPU. The SRR bit in this SRN is then reset to 0.

After sending the acknowledgement, the ICU resets ICR.PIPN to 0 and immediately starts a new arbitration process to determine if there is another pending interrupt request. If not, ICR.PIPN remains at 0 and the ICU enters an idle state, waiting for the next interrupt request to awaken it. If there is a new service request waiting, the priority number of the new request will be written to ICR.PIPN at the end of the new arbitration

process and the ICU will deliver the pending interrupt to the CPU according to the rules described in this section.

If a new service request is received by the ICU before the CPU has acknowledged the pending interrupt request, the ICU deactivates the pending request and starts a new arbitration process. This reduces the latency of service requests posted before the current request is acknowledged. The ICU deactivates the current pending interrupt request by setting the ICR.PIPN bit field to 0, indicating that the ICU has not yet found a new valid pending request. It then executes its arbitration process again. If the new service request has a higher priority than the previous one, its priority will be written to ICR.PIPN. If the new interrupt has a lower priority, the priority of the previous interrupt request will again be written to ICR.PIPN. In any case, the ICU will deliver a new interrupt request to the CPU according to the rules described in this section.

Once the CPU has acknowledged the current pending interrupt request, any new service request generated by an SRN must wait at least until the end of the next service request arbitration process to be serviced.

Essentially, arbitration in the ICU is performed whenever a new service request is detected, regardless of whether or not the CPU is servicing interrupts. Because of this, the ICR.PIPN bit field always reflects the pending service request with the highest priority. This can, for example, be used by software polling techniques to determine high-priority requests while leaving the interrupt system disabled.

14.5 Arbitration Process

The arbitration process implemented in the TC1765 uses a number of arbitration cycles to determine the pending interrupt request with the highest priority number, SRPN. In each of these cycles, two bits of the SRPNs of all pending service requests are compared against each other. The sequence starts with the high-order bits of the SRPNs and works downwards, such that in the last cycle, bits[1:0] of the SRPNs are compared. Thus, to perform an arbitration through all 8 bits of an SRPN, four arbitration cycles are required. There are two factors determining the duration of the arbitration process:

- Number of arbitration cycles, and
- Duration of arbitration cycles.

Both of these can be controlled by the user, as described in the following sections.

14.5.1 Controlling the Number of Arbitration Cycles

In a real-time system where responsiveness is critical, arbitration must be as fast as possible. Yet to maintain flexibility, the TC1765 system is designed to have a large range of service priorities. If not all priorities are needed in a system, arbitration can be speeded up by not examining all the bits used to identify all 255 unique priorities.

For instance, if a 6-bit number is enough to identify all priority numbers used in a system, (meaning that bits [7:6] of all SRPNs are always 0), it is not necessary to perform arbitration on these two bits. Three arbitration cycles will be enough to find the highest number in bits [5:0] of the SRPNs of all pending requests. Similarly, the number of arbitration cycles can be reduced to two if only bits [3:0] are used in all SRPNs, and the number of arbitration cycles can be reduced to one cycle if only bits [1:0] are used.

The ICR.CARBCYC bit field controls the number of cycles in the arbitration process. Its default value is 0, which selects four arbitration cycles. **Table 14-1** gives the options for arbitration cycle control.

Table 14-1 Arbitration Cycle Control

Number of Arbitration Cycles	4	3	2	1
ICR.CARBCYC	00_B	01_B	10_B	11_B
Relevant bits of the SRPNs	[7:0]	[5:0]	[3:0]	[1:0]
Range of priority numbers covered	1..255	1..63	1..15	1..3

Note: If less than four arbitration cycles are selected, the corresponding upper bits of the SRPNs are not examined, even if they do not contain zeros.

14.5.2 Controlling the Duration of Arbitration Cycles

During each arbitration cycle, the rate of information flow between the SRNs and the ICU can become limited by propagation delays within the TC1765 when it is executing at high system clock frequencies. At high frequencies, arbitration cycles may require two system clocks to execute properly. In order to optimize the arbitration scheme at lower system frequencies, an additional control bit, ICR.CONECYC is implemented. The default value of 0 of this bit selects two clock cycles per arbitration cycle. Setting this bit to 1 selects one clock cycle per arbitration cycle. This bit should only be set to 1 for lower system frequencies. Setting this bit for system frequencies above the specified limit leads to unpredictable behavior of the interrupt system. Correct operation is not then guaranteed.

14.6 Entering an Interrupt Service Routine

When an interrupt request from the ICU is pending and all conditions are met such that the CPU can now service the interrupt request, the CPU performs the following actions in preparation for entering the designated Interrupt Service Routine (ISR):

1. Upper context of the current task is saved[1)]. The current CPU priority number, ICR.CCPN, and the state of the global interrupt enable bit, ICR.IE, are automatically saved with the PCXI register (bit field PCPN and bit PIE).
2. Interrupt system is globally disabled (ICR.IE is set to 0).
3. Current CPU priority number (ICR.CCPN) is set to the value of ICR.PIPN.
4. PSW is set to a default value:
   - All permissions are enabled, that is, PSW.IO = 10_B.
   - Memory protection is switched to PRS0, that is, PSW.PRS = 0.
   - The stack pointer bit is set to the interrupt stack, that is, PSW.IS = 1.
   - The call depth counter is cleared, the call depth limit is set to 63, that is, PSW.CDC = 0.
5. Stack pointer, A10, is reloaded with the contents of the Interrupt Stack Pointer, ISP, if the PSW.IS bit of the interrupted routine was set to 0 (using the user stack), otherwise it is left unaltered.
6. CPU program counter is assigned an effective address consisting of the contents of the BIV register ORed with the ICR.PIPN number left-shifted by 5. This indexes the Interrupt Vector Table entry corresponding to the interrupt priority.
7. Contents at the effective address of the program counter in the Interrupt Vector Table is fetched as the first instruction of the Interrupt Service Routine (ISR). Execution continues linearly from there until the ISR branches or exits.

1) Note that, if a context-switch trap occurs while the CPU is in the process of saving the upper context of the current task, the pending ISR will not be entered, the interrupt request will be left pending, and the CPU will enter the appropriate trap handling routine instead.

As explained, receipt of further interrupts is disabled (ICR.IE = 0) when an Interrupt Service Routine is entered. At the same time, the current CPU priority ICR.CCPN is set by hardware to the priority of the interrupting source (ICR.PIPN).

Clearly, before the processor can receive any more interrupts, the ISR must eventually re-enable the interrupt system again by setting ICR.IE = 1. Furthermore, the ISR can also modify the priority number ICR.CCPN to allow effective interrupt priority levels. It is up to the user to enable the interrupt system again and optionally modify the priority number CCPN to implement interrupt priority levels or handle special cases (see next sections).

To simply enable the interrupt system again, the ENABLE instruction can be used, which sets ICR.IE bit to 1. The BISR instruction offers a convenient way to re-enable the interrupt system, to set ICR.CCPN to a new value, and to save the lower context of the interrupted task. It is also possible to use an MTCR instruction to modify ICR.IE and ICR.CCPN. However, this should be performed together with an ISYNC instruction (which synchronizes the instruction stream) to ensure completion of this operation before the execution of following instructions.

Note: The lower context can also be saved through execution of a SVLCX (Save Lower Context) instruction.

14.7 Exiting an Interrupt Service Routine

When an ISR exits with an RFE (Return From Exception) instruction, the hardware automatically restores the upper context. Register PCXI, which holds the Previous CPU Priority Number (PCPN) and the Previous Global Interrupt Enable Bit (PIE), is a part of this upper context. The value saved in PCPN is written to ICR.CCPN to set the CPU priority number to the value before the interruption, and bit PIE is written to ICR.IE to restore the state of this bit. The interrupted routine then continues.

Note: There is no automatic restoring of the lower context on an exit from an Interrupt Service Routine. If the lower context was saved during the execution of the ISR, either through execution of the BISR instruction or a SVLCX instruction, the ISR must restore the lower context again via the RSLCX (Restore Lower Context) instruction before it exits through RFI execution.

14.8 Interrupt Vector Table

Interrupt Service Routines are associated with interrupts at a particular priority by way of the Interrupt Vector Table. The Interrupt Vector Table is an array of Interrupt Service Routine entry points.

When the CPU takes an interrupt, it calculates an address in the Interrupt Vector Table that corresponds with the priority of the interrupt (the ICR.PIPN bit field). This address is loaded in the program counter. The CPU begins executing instructions at this address in the Interrupt Vector Table. The code at this address is the start of the selected Interrupt Service Routine (ISR). Depending on the code size of the ISR, the Interrupt Vector Table may only store the initial portion of the ISR, such as a jump instruction that vectors the CPU to the rest of the ISR elsewhere in memory.

The Interrupt Vector Table is stored in code memory. The BIV register specifies the base address of the Interrupt Vector Table. Interrupt vectors are ordered in the table by increasing priority.

The Base of Interrupt Vector Table register (BIV) stores the base address of the Interrupt Vector Table. It can be assigned to any available code memory. Its default on power-up is fixed at $0000\ 0000_H$. However, the BIV register can be modified using the MTCR instruction during the initialization phase of the system, before interrupts are enabled. With this arrangement, it is possible to have multiple Interrupt Vector Tables and switch between them by changing the contents of the BIV register.

*Note: The BIV register is protected by the ENDINIT bit (see **Chapter 18**). Modifications should only be done while the interrupt system is globally disabled (ICR.IE = 0). Also, an ISYNC instruction should be issued after modifying BIV to ensure completion of this operation before execution of following instructions.*

When interrupted, the CPU calculates the entry point of the appropriate Interrupt Service Routine from the PIPN and the contents of the BIV register. The PIPN is left-shifted by five bits and ORed with the address in the BIV register to generate a pointer into the Interrupt Vector Table. Execution of the ISR begins at this address. Due to this operation, it is recommended that bits [12:5] of register BIV are set to 0 (see **Figure 14-2**). Note that bit 0 of the BIV register is always 0 and cannot be written to (instructions have to be aligned on even byte boundaries).

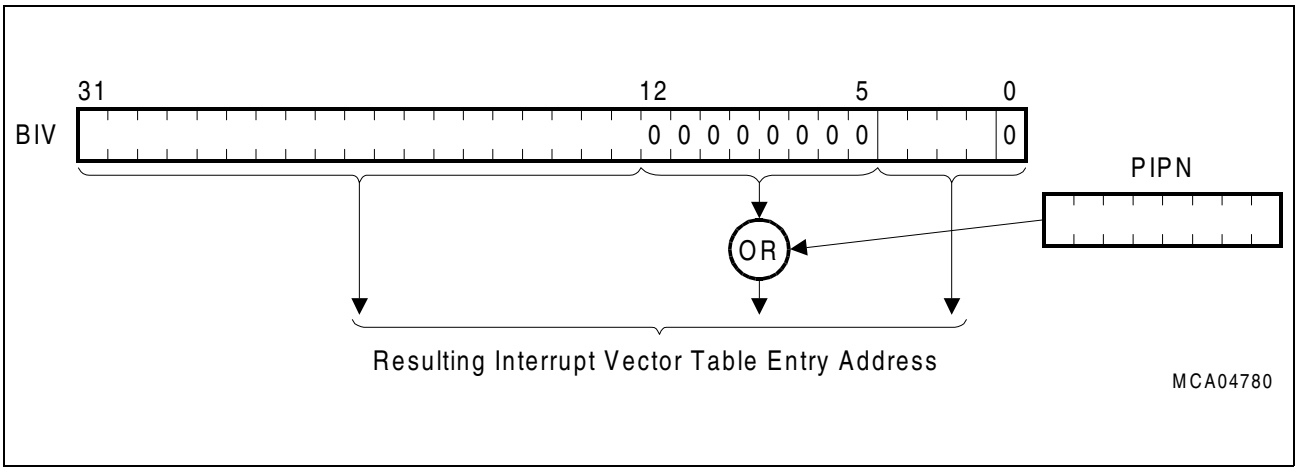


Figure 14-2 Interrupt Vector Table Entry Address Calculation

Left-shifting the PIPN by 5 bits creates entries into the Interrupt Vector Table which are evenly spaced 8 words apart. If an ISR is very short, it may fit entirely within the eight words available in the vector table entry. Otherwise, the code at the entry point must ultimately cause a jump to the rest of the ISR residing elsewhere in memory. Due to the way the vector table is organized according to the interrupt priorities, the TC1765 offers an additional option by allowing to span several Interrupt Vector Table entries so long as those entries are otherwise unused. **Figure 14-3** illustrates this.

The required size of the Interrupt Vector Table depends only on the range of priority numbers actually used in a system. Of the 256 vector entries, 255 may be used. Vector entry 0 is never used, because if ICR.PIPN is 0, the CPU is not interrupted. Distinct interrupt handlers are supported, but systems requiring fewer entries need not dedicate the full memory area required by the largest configurations.

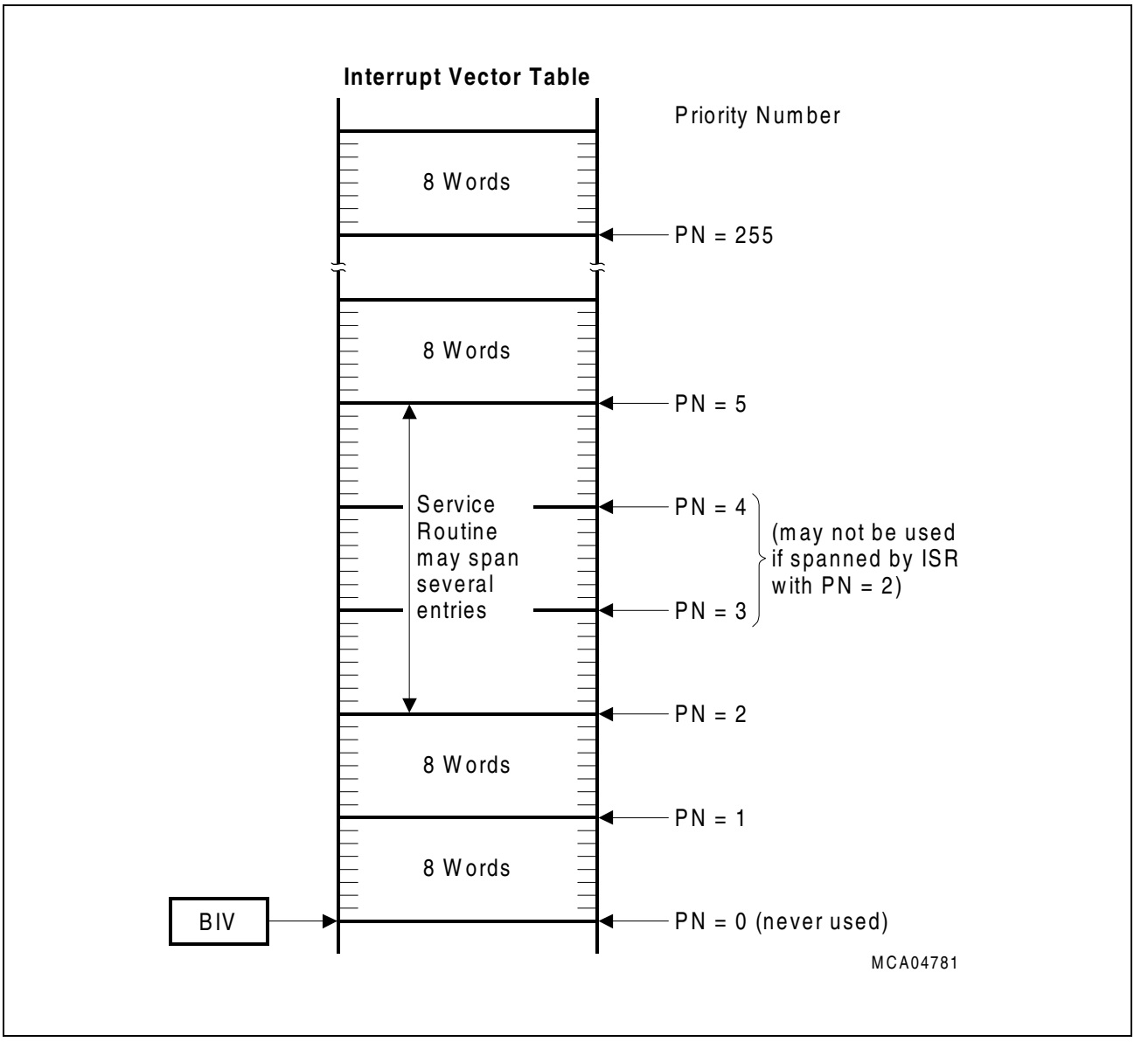


Figure 14-3 Interrupt Vector Table

14.9 Usage of the TC1765 Interrupt System

The following sections give some examples of using the TC1765 interrupt system to solve both typical and special application requirements.

14.9.1 Spanning Interrupt Service Routines Across Vector Entries

Each Interrupt Vector Table entry consists of eight words of memory. If an ISR can be made to fit directly in the Interrupt Vector Table there is no need for a jump instruction to vector to the rest of the interrupt handler elsewhere in memory. However, only the simplest ISRs can fit in the eight words available to a single entry in the table. But it is easy to arrange for ISRs to span across multiple entries, since the Interrupt Vector Table is ordered not by the interrupt source but by interrupt priority. This technique is explained in this section.

In the example of **Figure 14-3**, entry locations 3 and 4 are occupied by the ISR for entry 2. In **Figure 14-3**, the next available entry after entry 2 is entry 5. Of course, if this technique is used, it would be improper to allow any SRN to request service at any of the spanned vector priorities. Thus, priority levels 3 and 4 must not be assigned to SRNs requesting CPU service. They can, however, be used to request DMA service.

There is a performance trade-off which may arise when using this technique because the range of priority numbers used increases. This may have an impact on the number of arbitration cycles required to perform arbitration. Consider the case in which a system uses only three active interrupt sources, that is, where there are only three SRNs enabled to request service. If these three active sources are assigned to priority numbers 1, 2, and 3, it would be sufficient to perform the arbitration in just one cycle. However, if the ISR for interrupt priority 2 is spanned across three Interrupt Vector Table entries as shown in **Figure 14-3**, the priority numbers 1, 2 and 5 would have to be assigned. Thus, two arbitration cycles would have to be used to perform the full arbitration process.

The trade-off between the performance impact of the number of arbitration cycles and the performance gain through spanning service routines can be made by the system designer depending on system needs. Reducing the number of arbitration cycles reduces the service request arbitration latency - spanning service routines reduces the run time of service routines (and therefore also the latency for further interrupts at that priority level or below). For example, if there are multiple fleeting measurements to be made by a system, reducing arbitration latency may be most important. But if keeping total interrupt response time to a minimum is most urgent, spanning Interrupt Vector Table entries may be a solution.

14.9.2 Configuring Ordinary Interrupt Service Routines

When the CPU starts to service an interrupt, the interrupt system is globally disabled and the CPU priority ICR.CCPN is set to the priority of the interrupt now being serviced. This blocks all further interrupts from being serviced until the interrupt system is enabled again.

After an ordinary ISR begins execution, it is usually desirable for the ISR to re-enable global interrupts so that higher-priority interrupts (that is, interrupts that are greater than the current value of ICR.CCPN) can be serviced even during the current ISR's execution. Thus, such an ISR may set ICR.IE = 1 again with, for instance, the ENABLE instruction.

If the ISR enables the interrupt system again by setting ICR.IE = 1 but does not change ICR.CCPN, the effect is that from that point on the hardware can be interrupted by higher-priority interrupts but will be blocked from servicing interrupt requests with the same or lower priority than the current value of ISR.CCPN. Since the current ISR is clearly also at this priority level, the hardware is also blocked from delivering further interrupts to it as well. (This condition is clearly necessary so that the ISR can service the interrupt request atomically.)

When the ISR is finished, it exits with an RFE instruction. Hardware then restores the values of ICR.CCPN and ICR.IE to the values of the interrupted program.

14.9.3 Interrupt Priority Groups

It is sometimes useful to create groups of interrupts at the same or different interrupt priorities that cannot interrupt each other's ISRs. For instance, devices which can generate multiple interrupts, such as the General Purpose Timer, may need to have interrupts at different priorities interlocked in this way. The TC1765 interrupt architecture can be used to create such interrupt priority groups. It is effected by managing the current CPU priority level ICR.CCPN in a way described in this section.

If it is wished, for example, to make an interrupt priority group out of priority numbers 11 and 12, one would not want an ISR executing at priority 11 to be interrupted by a service request at priority 12, since this would be in the same priority group. One would wish that only interrupts above 12 should be allowed to interrupt the ISRs in this interrupt priority group. However, under ordinary ISR usage the ISR at priority 11 would be interrupted by any request with a higher priority number, including priority 12.

If, however, all ISRs in the interrupt priority group set the value of ICR.CCPN to the highest priority level within their group before they re-enable interrupts, then the desired interlocking will be effected.

Figure 14-4 shows an example for this. The interrupt requests with the priority numbers 11 and 12 form one group, while the requests with priority numbers 14 through 17 form another group. Each ISR in group 1 sets the value of ICR.CCPN to 12, the highest number in that group, before re-enabling the interrupt system. Each ISR in group 2 sets the value of ICR.CCPN to 17 before re-enabling the interrupt system. If, for example,

interrupt 14 is serviced, it can only be interrupted by requests with a priority number higher than 17; therefore it will not be interrupted by requests from its own priority group or requests with lower priority.

In **Figure 14-4**, the interrupt request with priority number 13 can be said to form an interrupt priority group with just itself as a member.

Setting ICR.CCPN to the maximum value 255 in each service routine has the same effect as not re-enabling the interrupt system; all interrupt requests can then be considered to be in the same group.

Interrupt priority groups are an example of the power of the TC1765 priority-based interrupt-ordering system. Thus the flexibility of interrupt priority levels ranges from all interrupts in one group to each interrupt request building its own group, and to all possible combinations in between.

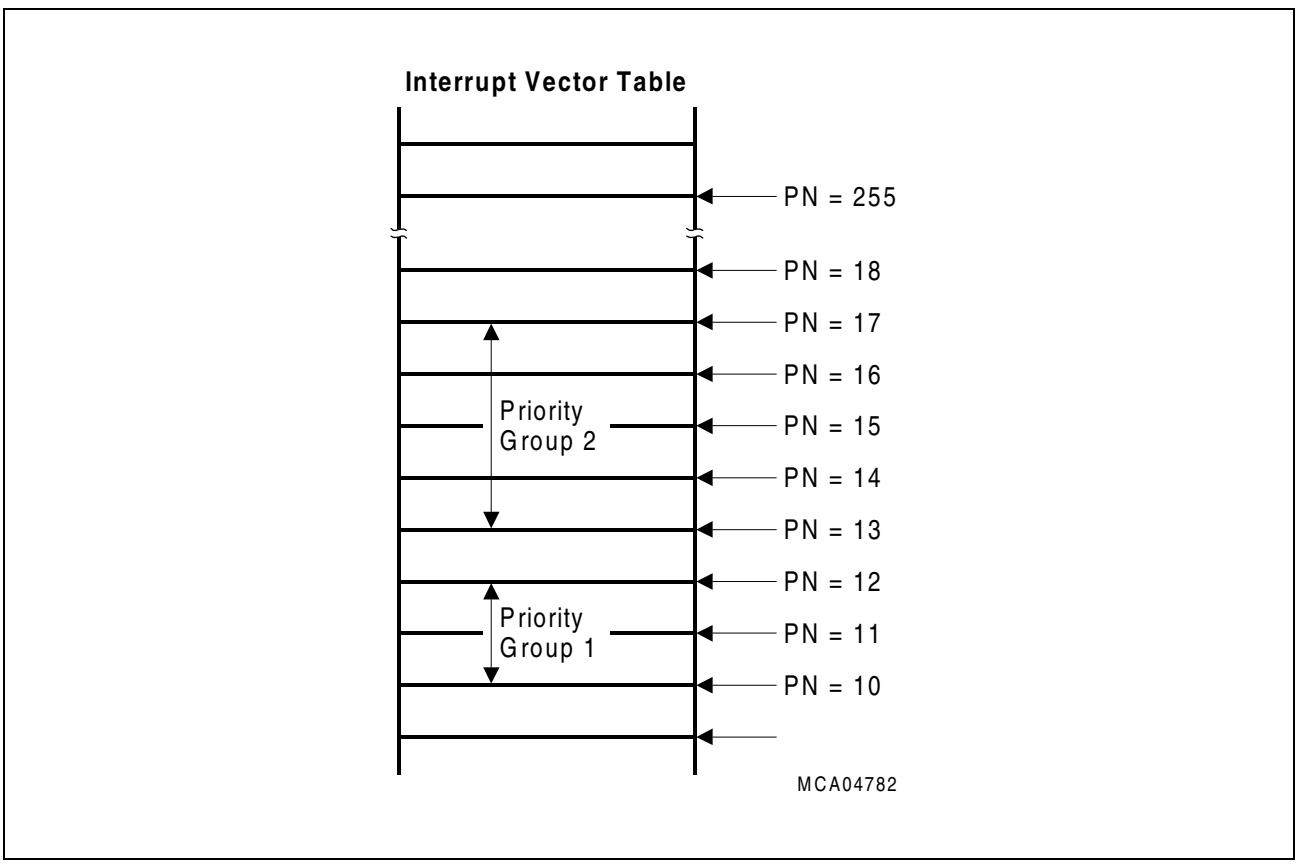


Figure 14-4 Interrupt Priority Groups

14.9.4 Splitting Interrupt Service Across Different Priority Levels

Interrupt service can be divided into multiple ISRs that execute at different priority levels. For example, the beginning stage of interrupt service may be very time-critical, such as to read a data value within a limited time window after the interrupt request activation. However, once the time-critical phase is past, there may still be more to do — for

instance, to process the observation. During this second phase, it might be acceptable for this ISR to be interrupted by lower-level interrupts. This can be performed as follows.

Say for example, the initial interrupt priority is fixed very high because response time is critical. The necessary actions are carried out immediately by the ISR at that high-priority level. Then the ISR prepares to invoke another ISR at a lower priority level through software to perform the lower-priority actions.

To invoke an ISR through software, the high-priority ISR directly sets an interrupt request bit in a SRN that will invoke the appropriate low-priority ISR. Then the high-priority ISR exits.

When the high-priority ISR exits, the pending low-priority interrupt will eventually be serviced (depending on the priority of other pending interrupts). When the low-priority ISR eventually executes, the low-priority actions of the interrupt will be performed.

The inverse of this method can also be employed, where a low-priority ISR raises its own priority level, or leaves interrupts turned off while it executes. For instance, the priority of a service request might be low because the time to respond to the event is not critical, but once it has been granted service, this service should not be interrupted. In this case, the ISR could raise the value of ICR.CCPN to a priority that would exclude some or all other interrupts, or simply leave interrupts disabled.

14.9.5 Using Different Priorities for the same Interrupt Source

For some applications, the urgency of a service request may vary, depending on the current state of the system. To handle this, different priority numbers (SRPNs) can be assigned at different times to a service request depending on the application needs.

Of course, Interrupt Service Routines must be placed in the Interrupt Vector Table at all addresses corresponding to the range of priorities used. If service remains the same at different priorities, copies of the ISR can be placed at the possible different entries, or the entries can all vector to a common ISR. If the ISR should execute different code depending on its priority, one need merely put the appropriate ISR in the appropriate entry of the Interrupt Vector Table.

This flexibility is another advantage of the TC1765 interrupt architecture. In traditional interrupt systems where the interrupt vectors are ordered by interrupting source, the ISR would have to check the current priority of the interrupt request and perform a branch to the appropriate code section, causing a delay in the response to the request. In the TC1765, however, the extra check and branch in the ISR are not necessary, which reduces the interrupt latency.

Because this approach may necessitate an increase in the range of interrupt priorities, the system designer must trade off this advantage against any possible increase in the number of arbitration cycles.

14.9.6 Software Initiated Interrupts

Software can set the service request bit in a SRN by writing to its Service Request Control Register. Thus, software can initiate interrupts which are handled by the same mechanism as hardware interrupts.

After the service request bit is set in an active SRN, there is no way to distinguish between a software initiated interrupt request and a hardware interrupt request. For this reason, software should only use SRNs and interrupt priority numbers that are not being used for hardware interrupts.

The TC1765 architecture includes four Service Request Nodes which are intended solely for the purpose of generating software interrupts. These SRNs are not connected to any hardware that could generate a service request, and so are only able to be used by software. Additionally, any otherwise unused SRN can be employed to generate software interrupts.

14.9.7 Interrupt Priority 1

Interrupt Priority 1 is the first and lowest-priority entry in the Interrupt Vector Table. It is generally reserved for ISRs which perform task management. ISRs whose actions cause software-managed tasks to be created post a software interrupt request at priority level 1 to signal the event.

The ISR that triggers this event can then execute a normal return from interrupt. There is no need for it to check whether the ISR is returning to the background-task priority level (priority 0) or is returning to a lower-priority ISR that it interrupted. When there is a pending interrupt at a priority higher than the return context for the current interrupt, this interrupt will then be serviced. When a return to the background-task priority level (level 0) is performed, the software-posted interrupt at priority level 1 will be serviced automatically.

14.10 CPU Service Request Nodes

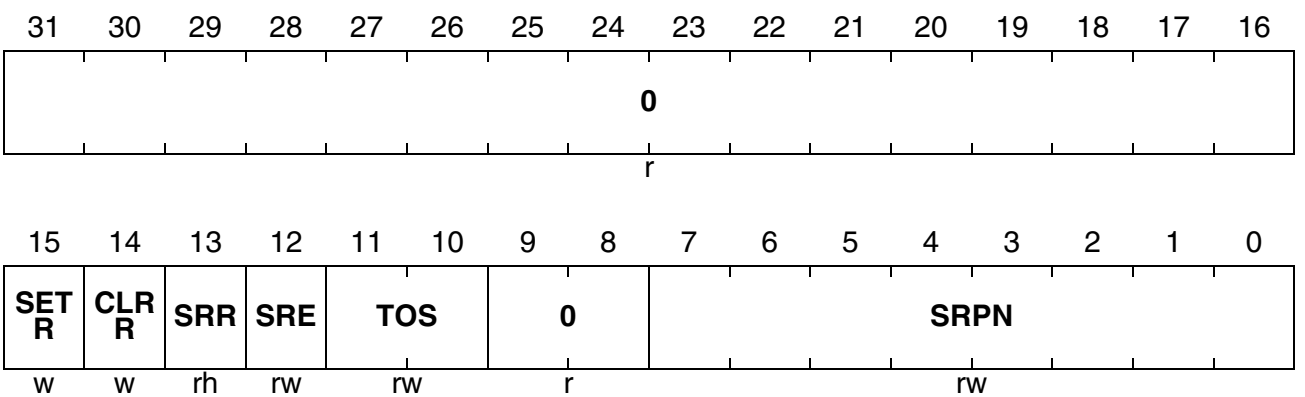
To support software initiated interrupts, the TC1765 contains four Service Request Nodes which are not attached to triggering peripherals. These SRNs can only cause interrupts when software sets the service request bit in one of their Service Request Control Registers. These SRNs are called the CPU Service Request Nodes.

Additionally, any otherwise unused SRN can be employed to generate software interrupts.

Note: The CPU Service Request Control Registers are not bit-addressable.

CPU_SRC0
CPU Service Request Control Register 0
CPU_SRC1
CPU Service Request Control Register 1
CPU_SRC2
CPU Service Request Control Register 2
CPU_SRC3
CPU Service Request Control Register 3

Reset Values: 0000 0000$_H$



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	[11:10]	rw	**Type of Service Control;** must be written with 00_B.
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], [31:16]	r	**Reserved**

*Note: See **Section 14.3.1** for detailed register description.*

14.11 Service Request Register Table

Table 14-2 shows all SRN registers with its short and long name.

Table 14-2 Service Request Control Register Table

Short Name	Long Name
Bus Control Unit (BCU)	
BCU_SRC	BCU Service Request Control Register
General Purpose Timer Unit (GPTU)	
GPTU_SRC0	GPTU Service Request Control Register 0
GPTU_SRC1	GPTU Service Request Control Register 1
GPTU_SRC2	GPTU Service Request Control Register 2
GPTU_SRC3	GPTU Service Request Control Register 3
GPTU_SRC4	GPTU Service Request Control Register 4
GPTU_SRC5	GPTU Service Request Control Register 5
GPTU_SRC6	GPTU Service Request Control Register 6
GPTU_SRC7	GPTU Service Request Control Register 7
Asynchronous Serial Channels (ASC0/ASC1)	
ASC0_TSRC	ASC0 Transmit Interrupt Service Request Control Register
ASC0_RSRC	ASC0 Receive Interrupt Service Request Control Register
ASC0_ESRC	ASC0 Error Interrupt Service Request Control Register
ASC0_TBSRC	ASC0 Transmit Buffer Interrupt Service Req. Control Register
ASC1_TSRC	ASC1 Transmit Interrupt Service Request Control Register
ASC1_RSRC	ASC1 Receive Interrupt Service Request Control Register
ASC1_ESRC	ASC1 Error Interrupt Service Request Control Register
ASC1_TBSRC	ASC1 Transmit Buffer Interrupt Service Req. Control Register
Synchronous Serial Channels (SSC0/SSC1)	
SSC0_TSRC	SSC0 Transmit Interrupt Service Request Control Register
SSC0_RSRC	SSC0 Receive Interrupt Service Request Control Register
SSC0_ESRC	SSC0 Error Interrupt Service Request Control Register
SSC1_TSRC	SSC1 Transmit Interrupt Service Request Control Register
SSC1_RSRC	SSC1 Receive Interrupt Service Request Control Register
SSC1_ESRC	SSC1 Error Interrupt Service Request Control Register

Table 14-2 Service Request Control Register Table (cont'd)

Short Name	Long Name
General Purpose Timer Array (GPTA)	
GPTA_SRC00	GPTA Service Request Control Register 00
GPTA_SRC01	GPTA Service Request Control Register 01
GPTA_SRC02	GPTA Service Request Control Register 02
GPTA_SRC03	GPTA Service Request Control Register 03
GPTA_SRC04	GPTA Service Request Control Register 04
GPTA_SRC05	GPTA Service Request Control Register 05
GPTA_SRC06	GPTA Service Request Control Register 06
GPTA_SRC07	GPTA Service Request Control Register 07
GPTA_SRC08	GPTA Service Request Control Register 08
GPTA_SRC09	GPTA Service Request Control Register 09
GPTA_SRC10	GPTA Service Request Control Register 10
GPTA_SRC11	GPTA Service Request Control Register 11
GPTA_SRC12	GPTA Service Request Control Register 12
GPTA_SRC13	GPTA Service Request Control Register 13
GPTA_SRC14	GPTA Service Request Control Register 14
GPTA_SRC15	GPTA Service Request Control Register 15
GPTA_SRC16	GPTA Service Request Control Register 16
GPTA_SRC17	GPTA Service Request Control Register 17
GPTA_SRC18	GPTA Service Request Control Register 18
GPTA_SRC19	GPTA Service Request Control Register 19
GPTA_SRC20	GPTA Service Request Control Register 20
GPTA_SRC21	GPTA Service Request Control Register 21
GPTA_SRC22	GPTA Service Request Control Register 22
GPTA_SRC23	GPTA Service Request Control Register 23
GPTA_SRC24	GPTA Service Request Control Register 24
GPTA_SRC25	GPTA Service Request Control Register 25
GPTA_SRC26	GPTA Service Request Control Register 26
GPTA_SRC27	GPTA Service Request Control Register 27
GPTA_SRC28	GPTA Service Request Control Register 28

Table 14-2 Service Request Control Register Table (cont'd)

Short Name	Long Name
GPTA_SRC29	GPTA Service Request Control Register 29
GPTA_SRC30	GPTA Service Request Control Register 30
GPTA_SRC31	GPTA Service Request Control Register 31
GPTA_SRC32	GPTA Service Request Control Register 32
GPTA_SRC33	GPTA Service Request Control Register 33
GPTA_SRC34	GPTA Service Request Control Register 34
GPTA_SRC35	GPTA Service Request Control Register 35
GPTA_SRC36	GPTA Service Request Control Register 36
GPTA_SRC37	GPTA Service Request Control Register 37
GPTA_SRC38	GPTA Service Request Control Register 38
GPTA_SRC39	GPTA Service Request Control Register 39
GPTA_SRC40	GPTA Service Request Control Register 40
GPTA_SRC41	GPTA Service Request Control Register 41
GPTA_SRC42	GPTA Service Request Control Register 42
GPTA_SRC43	GPTA Service Request Control Register 43
GPTA_SRC44	GPTA Service Request Control Register 44
GPTA_SRC45	GPTA Service Request Control Register 45
GPTA_SRC46	GPTA Service Request Control Register 46
GPTA_SRC47	GPTA Service Request Control Register 47
GPTA_SRC48	GPTA Service Request Control Register 48
GPTA_SRC49	GPTA Service Request Control Register 49
GPTA_SRC50	GPTA Service Request Control Register 50
GPTA_SRC51	GPTA Service Request Control Register 51
GPTA_SRC52	GPTA Service Request Control Register 52
GPTA_SRC53	GPTA Service Request Control Register 53
Analog-to-Digital Converter (ADC0/ADC1)	
ADC0_SRC0	ADC0 Service Request Control Register 0
ADC0_SRC1	ADC0 Service Request Control Register 1
ADC0_SRC2	ADC0 Service Request Control Register 2
ADC0_SRC3	ADC0 Service Request Control Register 3

Table 14-2 Service Request Control Register Table (cont'd)

Short Name	Long Name
ADC1_SRC0	ADC1 Service Request Control Register 0
ADC1_SRC1	ADC1 Service Request Control Register 1
ADC1_SRC2	ADC1 Service Request Control Register 2
ADC1_SRC3	ADC1 Service Request Control Register 3
DMA Controller (DMA)	
DMA_SRC0	DMA Service Request Control Register 0
DMA_SRC1	DMA Service Request Control Register 1
DMA_SRC2	DMA Service Request Control Register 2
DMA_SRC3	DMA Service Request Control Register 3
DMA_SRC4	DMA Service Request Control Register 4
DMA_SRC5	DMA Service Request Control Register 5
DMA_SRC6	DMA Service Request Control Register 6
DMA_SRC7	DMA Service Request Control Register 7
Controller Area Network Module (CAN)	
CAN_SRC0	CAN Service Request Control Register 0
CAN_SRC1	CAN Service Request Control Register 1
CAN_SRC2	CAN Service Request Control Register 2
CAN_SRC3	CAN Service Request Control Register 3
CAN_SRC4	CAN Service Request Control Register 4
CAN_SRC5	CAN Service Request Control Register 5
CAN_SRC6	CAN Service Request Control Register 6
CAN_SRC7	CAN Service Request Control Register 7
OCDS	
SBSRC0	Software Break Service Request Control Register 0
CPU	
CPU_SRC0	CPU Service Request Control Register 0
CPU_SRC1	CPU Service Request Control Register 1
CPU_SRC2	CPU Service Request Control Register 2
CPU_SRC3	CPU Service Request Control Register 3

15 Trap System

The TC1765 trap system provides a means for the CPU to service conditions that are so critical that they must not be postponed. Such conditions include both catastrophic developments, such as an attempt by the CPU to execute an illegal instruction, as well as routine developments such as system calls. This chapter describes the trap system for the TC1765. Topics covered include trap types, trap handling, and non-maskable interrupts (NMIs). Traps direct the processor to execute Trap Service Routines (TSR) stored in a Trap Vector Table.

15.1 Trap System Overview

Traps break the normal execution of code, much like interrupts, but traps are different from interrupts in these ways:

- TSRs reside in the Trap Vector Table, which is separate from the Interrupt Vector Table.
- A trap does not change the CPU's interrupt priority, so the ICR.CCPN field is not changed.
- Traps cannot be disabled by software. Traps are always active.
- The return address saved when a Trap Service Routine is invoked is the address of the instruction in progress at the moment the trap was raised, whereas the return address of an interrupt is the address of the instruction that would have been executed next if the interrupt had not occurred.

The CPU aborts the instruction in progress when a trap occurs, and forces execution to the appropriate TSR. The TSR decides whether the situation is correctable or not. If not, the TSR takes appropriate action, which may involve aborting the current task, or even resetting the TC1765. If the situation is routine or correctable, the TSR performs whatever action is necessary, then exits, whereupon the CPU re-executes the previously aborted instruction.

Traps may arise within the CPU, for instance, as a side-effect of the execution of instructions. These traps are typically synchronous with the processor instruction clock. They may also be generated by events external to the CPU, such as a peripheral or external NMI signal. Hardware-generated traps are typically asynchronous with the processor instruction clock.

Traps can signal a variety of routine or serious events. For instance, traps can be used to

- Implement memory protection and virtual memory
- Provide unprivileged applications access to privileged system services
- Manage task-based context-switching
- Respond to urgent external conditions, such as an NMI
- Respond to urgent internal conditions, such as signals from the Watchdog Timer, the FPI Bus, or the PLL
- Detect access to memory by other system components

- Signal events from task to task
- Administer overflow and underflow of hardware tables and lists
- Recover from catastrophic software errors

Many traps arise as a consequence of the execution of instructions:

- The SYSCALL instruction generates a trap that is usually intended to signal a request for system services by an unprivileged application.
- An attempt to execute an illegal instruction opcode produces a trap as a side-effect. The instruction is aborted, and a trap is invoked. This protects a system from poorly-written or damaged programs.
- When an application attempts to execute an unimplemented instruction opcode, the trap that results can invoke a TSR to emulate the operation of that instruction in software, thereby extending the instruction set.
- If an application attempts to access protected memory, the resulting trap may be used by the system to read in pages from memory that the application needs.
- If an arithmetic operation produces an invalid result, a trap is generated. In some cases, the TSR may attempt to correct the result through software, or it may cause the application to terminate.

Other uses of traps include:

- Context management
- Recovery from FPI Bus error signals
- Access to memory by a peripheral
- Handling the Non-Maskable “Interrupt” (actually trap) signal from the external NMI input, from the Watchdog Timer, or from the PLL if it loses stable clock signals

When a hardware trap condition is detected, the processor’s trap control system supplies a two-part number that identifies the cause of the trap. The first part of the number is a three-bit Trap Class Number (TCN); the second part is an eight-bit Trap Identification Number (TIN). The TCN is used to index the Trap Vector Table to identify the proper TSR to handle the trap. The TIN is loaded into register D15 of the TSR’s context to further identify the precise cause of the trap. The TSR must examine the TIN in software.

15.2 Trap Classes

The TC1765 has eight trap classes, as shown in Table 15-1. Each trap has a Trap Identification Number (TIN), that identifies the number of the trap within its class. When the CPU hardware goes to service a trap, the TIN is loaded into register D15 before the first instruction of the trap handler is executed. A trap is completely identified by its Trap Class Number (TCN) and its TIN.

Table 15-1 summarizes and classifies all TC1765-supported traps. In the column “Type”, an “S” stands for a Synchronous trap, while “A” indicates an Asynchronous trap. “SW” and “HW” indicate a Software trap or a Hardware trap, respectively. The column “Saved PC” states which Program Counter value is saved during the trap entry. “ThisPC” indicates that the PC value of the instruction causing the trap is saved, while “NextPC” is the PC value pointing to the instruction which would have been executed next.

Table 15-1 TC1765 Supported Traps

Trap ID (TIN)	Trap Name	Trap Type	Saved PC	Description
Class 0 - Reset				
0	–	S, HW	–	Reserved
Class 1 - Internal Protection Traps (TCN = 1)				
1	PRIV	S, HW	ThisPC	Privileged Instruction
2	MPR	S, HW	ThisPC	Memory Protection: Read Access
3	MPW	S, HW	ThisPC	Memory Protection: Write Access
4	MPX	S, HW	ThisPC	Memory Protection: Execution Access
5	MPP	S, HW	ThisPC	Memory Protection: Peripheral Access
6	MPN	S, HW	ThisPC	Memory Protection: Null Address
7	GRWP	S, HW	ThisPC	Global Register Write Protection
Class 2 - Instruction Errors (TCN = 2)				
1	IOPC	S, HW	ThisPC	Illegal Opcode
4	ALN	S, HW	ThisPC	Data Address Alignment Error
5	MEM	S, HW	ThisPC	Invalid Memory Address
Class 3 - Context Management (TCN = 3)				
1	FCD	S, HW	ThisPC	Free Context List Depleted (FCX == LCX)
2	CDO	S, HW	ThisPC	Call Depth Overflow
3	CDU	S, HW	ThisPC	Call Depth Underflow
4	FCU	S, HW	see Note	Free Context List Underflow (FCX == 0)

Table 15-1 TC1765 Supported Traps (cont'd)

Trap ID (TIN)	Trap Name	Trap Type	Saved PC	Description
5	CSU	S, HW	ThisPC	Context List Underflow (PCX == 0)
6	CTYP	S, HW	ThisPC	Context Type Error (PCXI.UL is wrong)
7	NEST	S, HW	ThisPC	Nesting Error: RFE with non-zero call depth
Class 4 - System Bus Errors (TCN = 4)				
1	PSE	S, HW	ThisPC	Bus Error on Program Fetch Operation
2	DSE	S, HW	ThisPC	Bus Error on Data Load Operation
3	DAE	A, HW	ThisPC	Bus Error on Data Store Operation
Class 5 - Assertion Traps (TCN = 5)				
1	OVF	S, SW	ThisPC	Arithmetic Overflow
2	SOVF	S, SW	ThisPC	Sticky Arithmetic Overflow
Class 6 - System Call (TCN = 6)				
1)	SYS	S, SW	NextPC	System Call
Class 7 - Non-Maskable Interrupt (TCN = 7)				
0	NMI	A, HW	NextPC	Non-Maskable Interrupt

1) For the system call trap via the SYSCALL instruction, the TIN is created from an immediate constant in the SYSCALL instruction supplied by the calling software. The range of values for this constant is 0 to 255, inclusive. The normal trap entry mechanism is not used, instead, a jump to the FCU trap handler is performed.

15.2.1 Synchronous Traps

Synchronous traps are associated with the execution, or attempted execution, of processor instructions. The trap is taken immediately and serviced before execution can proceed beyond that instruction (except for the SYSCALL instruction).

15.2.2 Asynchronous Traps

Asynchronous traps are similar to interrupts, in that they are associated with hardware conditions detected externally and signaled back to the processor. Some asynchronous traps result indirectly from instructions that have been executed previously, but the direct association with those instructions has been lost. Others such as the NMI arise strictly from external events.

Note: Due to a missing trap queue in the TriCore architecture, it is possible to lose asynchronous traps (e.g. caused by an FPI Bus write operation) if several traps are generated within a very short time frame.

15.2.3 Hardware Traps

Hardware traps are generated as a result of problems encountered while executing processor instructions. Examples include attempting to execute an illegal instruction opcode, attempting to access protected memory, and attempting to access data memory at a misaligned address.

15.2.4 Software Traps

Software traps are used to make system calls and test assertions in software. For example, a client application can call a privileged system function by executing the SYSCALL instruction, which invokes a TSR to begin executing in privileged mode.

There is a single entry in the Trap Vector Table for the SYSCALL trap. An application executing the SYSCALL instruction must embed a system-defined eight-bit immediate constant in the SYSCALL instruction, which becomes the TIN for the SYSCALL trap. Thus the application can signal its need for specific privileged services.

15.2.5 Trap Descriptions

In this section, each of the traps listed in **Table 15-1** is described in more detail.

RESET Trap

This trap is not used in the TC1765.

PRIV Trap

The PRIV trap is detected in the decode stage of the load/store pipeline. The PRIV trap is generated whenever an attempt is made to execute a protected system instruction in User Mode. The protected system instructions are:

- MTCR
- BISR

A PRIV trap is also taken whenever an attempt is made to execute one of the following instructions in User Mode 0:

- ENABLE
- DISABLE

MPR Trap

Read memory protection violations are detected in the execute stage of the load/store pipeline. The MPR trap is generated for LD/LDMST and SWAP instructions when the memory protection system is enabled and the effective address does not lie within any range with read permissions enabled.

MPW Trap

Write memory protection violations are detected in the execute stage of the load/store pipeline. The MPW trap is generated for ST/LDMST and SWAP instructions when the memory protection system is enabled and the effective address does not lie within any range with read permissions enabled.

MPX Trap

The Execution Access Memory Protection Trap is detected in the fetch stage. The MPX trap is generated when the memory protection system is enabled and the PC does not lie within any range with execute permissions enabled.

MPP Trap

The Peripheral Access Memory Protection Trap is detected in the execute stage of the load/store pipeline. It is generated when either segment 14 or 15 is targeted by any memory operation while the machine is in User Mode 0.

MPN Trap

The Null Address Memory Protection Trap is detected in the execute stage of the load/store pipeline. It is generated when any memory operation targets address 0.

GRWP Trap

The GRWP trap is detected in the decode stage of the load/store pipeline. The GRWP trap is generated whenever an attempt is made to execute an instruction that modifies one of the four global registers, A0, A1, A8 and A9, while the Global Write Enable (PSW.GW) is 0.

IOPC Trap

The IOPC trap can be detected in either the integer or load/store decode stages. The IOPC trap is raised when an invalid instruction is decoded, that is, the instruction in the decode stage does not map onto a known opcode.

ALN Trap

The ALN trap is detected in the execute stage of the load/store pipeline. The trap is raised whenever a memory operation does not conform to the expected memory alignment constraints.

MEM Trap

The MEM trap is detected in the execute stage of the load/store pipeline. The trap is raised whenever an attempt is made to access an invalid memory address such as:

- An effective address that lies in a different segment to the base address
- An address that crosses a segment boundary
- An address range in the DMU or PMU that does not map onto an implemented area of memory
- An address in the Core SFRs (CSFRs)

FCD Trap

The FCD trap is detected in the decode stage of the load/store unit. An FCD trap is raised whenever a save context operation is performed and the Free Context List Pointer (FCX) equals the contents of the Free Context Limit Pointer (LCX).

CDO Trap

The CDO trap is detected in the decode stage of the load/store pipeline. The trap results when a call is attempted and the call-depth limit has been reached (call-depth counter overflow).

CDU Trap

The CDU trap is detected in the decode stage of the load/store pipeline. The trap results when a RET instruction is attempted and the call-depth counter equals 0.

FCU Trap

The Free Context List Underflow Trap is one of the most serious error conditions in the machine. The trap results when a save context operation is performed and the FCX equals 0. This trap is also raised if any error occurs during a context save operation.

The normal trap entry mechanism is not taken, instead a jump to the FCU trap handler is performed.

CSU Trap

The CSU trap is detected in the decode stage of the load/store pipeline. The trap results when a restore-context operation is performed and Previous Context Pointer (PCXI.PCX) equals 0.

CTYP Trap

The CTYP trap is detected in the decode stage of the load/store pipeline. The trap is raised when a context-restore operation is performed on an incorrect context type. That is if a restore lower context is performed when the PCXI.UL = 1, or a restore upper context is performed when the PCXI.UL = 0.

NEST Trap

The Nesting Error Trap (is detected in the decode stage of the load/store pipeline. The NEST trap results when an RFE instruction is attempted and the call depth counter does not equal 0.

PSE Trap

The Program Fetch Synchronous Error Trap is detected in the integer or load/store decode stage. The PSE trap is raised when the fetch of an instruction from the Program Memory Unit (PMU) results in an error (e.g. fetch from a reserved address).

DSE Trap

The Data Load/Store Synchronous Error Trap is detected in the execute stage of the load/store unit. The DSE trap is generated by the DMU on a cache management error, DMU control register access error, FPI Bus access error, or a DMU memory range error. The exact cause of the error can be read in the DMU Synchronous Trap Flag Register, DMU_STR. DSE traps occur in general on load accesses to the DMU.

DAE Trap

The Data Load/Store Asynchronous Error Trap is an asynchronous trap. The DSE trap is generated by the DMU on either a cache management error, DMU control register access error, FPI Bus access error, or a DMU memory range error. The exact cause of the error can be read via the DMU Asynchronous Trap Flag Register, DMU_ATR. DAE traps occur in general on store accesses to the DMU.

OVF Trap

The OVF trap is detected in the execute stage of the load/store pipeline. The trap is raised by the TRAPV instruction when the instruction is executed and the Overflow Flag, PSW.V, is set.

SOVF Trap

The SOVF trap is detected in the execute stage of the load/store pipeline. The trap is raised by the TRAPSV instruction when the instruction is executed and the Sticky Overflow Flag, PSW.SV, is set.

SYS Trap

The SYS trap is detected in the decode stage of the load/store pipeline. The trap is raised implicitly by the SYSCALL instruction. For the system call trap via the SYSCALL instruction, the TIN is created from an immediate constant in the SYSCALL instruction supplied by the calling software. The range of values for this constant is 0 through 255.

NMI Trap

The NMI is an asynchronous trap. The generation of the NMI is handled by the Power-Watchdog-Reset (PWR) block in the system. The source can be the external $\overline{\text{NMI}}$ input, a Watchdog Timer error condition, or a loss of stable clock signal in the PLL.

15.3 Trap Vector Table

The entry-points of all Trap Service Routines are stored in code memory in the Trap Vector Table. The BTV register specifies the base address of the Trap Vector Table in code memory. It can be assigned to any available code memory. Its default on power-up is fixed at A000 0100_H. However, the BTV register can be modified using the MTCR instruction during the initialization phase of the system. With this arrangement, it is possible to have multiple Trap Vector Tables and switch between them by changing the contents of the BTV register.

Note: The BTV register is protected by the ENDINIT bit. An ISYNC instruction should be issued after modifying BTV so as to avoid untoward pipeline behavior.

When a trap event occurs, a trap identifier is generated by the hardware detecting the event. The trap identifier is made up of a Trap Class Number (TCN) and a Trap Identification Number (TIN).

The TCN is left-shifted by five bits and ORed with the address in the BTV register to form the entry address of the TSR. Due to this operation, it is recommended that bits [7:5] of register BTV are set to 0 (see **Figure 15-1**). Note that bit 0 of the BTV register is always 0 and can not be written to (instructions have to be aligned on even byte boundaries).

Left-shifting the TCN by 5 bits creates entries into the Trap Vector Table which are evenly spaced 8 words apart. If a trap handler (TSR) is very short, it may fit entirely within the eight words available in the Trap Vector Table entry. Otherwise, the code at the entry point must ultimately cause a jump to the rest of the TSR residing elsewhere in memory.

Unlike the Interrupt Vector Table, entries in the Trap Vector Table cannot be spanned.

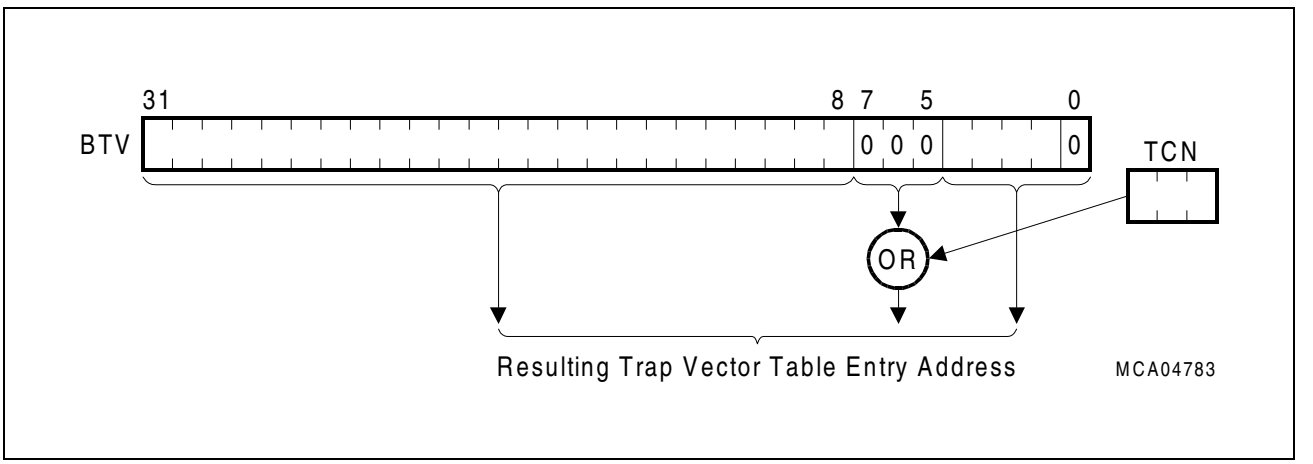


Figure 15-1 Trap Vector Table Entry Address Calculation

15.3.1 Entering a Trap Service Routine

The following actions are performed to enter a TSR when a trap event is detected by the hardware:

1. The upper context of the current task is saved[1)].
2. The interrupt system is globally disabled (ICR.IE = 0).
3. The current CPU priority number (CCPN) is not changed.
4. The PSW is set to a default value:
   - All permissions are enabled: PSW.IO = 10_B
   - Memory protection is switched to PRS 0: PSW.PRS = 00_B.
   - The stack pointer bit is set for using the interrupt stack: PSW.IS = 1.
   - The call-depth counter is cleared, the call depth limit is set for 64: PSW.CDC = 0.
5. The stack pointer, A10, is reloaded with the contents of the Interrupt Stack Pointer, ISP, if the PSW.IS bit of the interrupted routine was set to 0 (using the user stack), otherwise it is left unaltered.
6. The Trap Vector Table is accessed to fetch the first instruction of the TSR. The effective address is the contents of the BTV register ORed with the Trap Class Number (TCN) left-shifted by 5.

Although traps leave the ICR.CCPN unchanged, TSRs still begin execution with interrupts disabled. They can therefore perform critical initial operations without interruption, until they specifically re-enable interrupts.

Since entry into a trap handler is only determined by the TCN, software in the TSR must determine the exact cause of the trap by evaluation of the TIN stored in register D15.

1) If a context-switch trap occurs while the CPU is in the process of saving the upper context of the current task, the pending ISR will not be entered, the interrupt request will be left pending, and the CPU will enter the appropriate trap handling routine instead.

15.4 Non-Maskable Interrupt

Although called an interrupt, the non-maskable interrupt (NMI) is actually serviced as a trap, since it is not interruptible and does not follow the standards for regular interrupts.

In the TC1765, three different events can generate an NMI trap:

- A transition on the $\overline{\text{NMI}}$ input pin
- An error or wake-up signal from the Watchdog Timer
- The PLL upon loss of external clock stability

The type of an NMI trap is indicates in the NMI Status Register (NMISR).

15.4.1 NMI Status Register

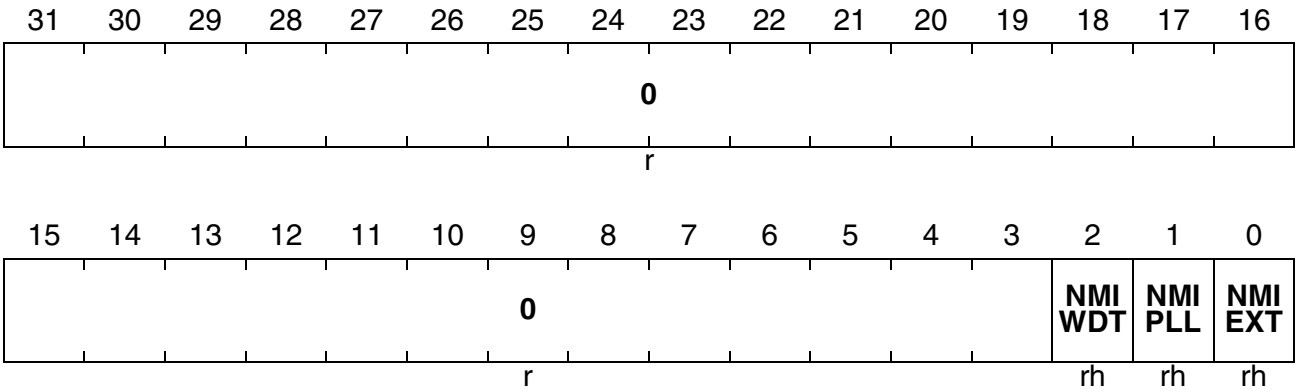
The source of an NMI trap can be identified through three status bits in NMISR. The bits in NMISR are read-only; writing to them has no effect.

The CPU detects a one-to-zero transition of the $\overline{\text{NMI}}$ input signal as indicating a NMI trap event. It then sets NMISR.NMIEXT. If the Watchdog Timer times out, it sets NMISR.NMIWDT. If the PLL loses its clock signal, it sets NMISR.PLL.

The bits in NMISR are ORed together to generate an NMI trap request to the CPU. If one of the NMISR bits is newly asserted while another bit is set, no new NMI trap request is generated. All flags are cleared automatically after a read of NMISR. Therefore, after reading NMISR, the NMI TSR must check all bits in NMISR to determine whether there have been multiple causes of an NMI trap.

Note: The NMISR register is located in the address range reserved for the System Control Unit (SCU).

NMISR
NMI Status Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
NMIEXT	0	rh	**External NMI Flag** 0 No external NMI request has occurred 1 An external NMI request has been detected *Note: This bit is reset after an exit from Deep Sleep Mode through pin $\overline{NMI}$ when bit PMG_CON.DSRW has been set before (full reset sequence executed).*
NMIPLL	1	rh	**PLL NMI Flag** 0 No PLL NMI has occurred 1 The PLL has lost the lock to the external crystal
NMIWDT	2	rh	**Watchdog Timer NMI Flag** 0 No watchdog NMI occurred 1 The Watchdog Timer has entered the pre-warning phase due to a watchdog error
0	[31:3]	r	**Reserved**; read as 0.

15.4.2 External $\overline{NMI}$ Input

An external NMI event is generated when a one-to-zero transition is detected at the external $\overline{NMI}$ input pin. NMISR.NMIEXT is set in this case. The $\overline{NMI}$ pin is sampled at the system clock frequency. A transition is recognized when one sample shows a 1 and the next sample shows a 0. Subsequent 0-samples or a 0-to-1 transition do not trigger any action.

15.4.3 Phase-Locked Loop NMI

The PLL clock generation unit sets the NMIPLL flag when it detects a loss in the synchronization with the external oscillator clock input. This condition means that the

PLL clock frequency is no longer stable, and that the PLL will now decrease to its base frequency.

15.4.4 Watchdog Timer NMI

The Watchdog Timer sets the NMIWDT flag for two conditions:

- A Watchdog Timer error has occurred
- Bit 15 of the Watchdog Timer is set while the CPU is in idle mode

A Watchdog Timer error can produce an NMI event because

- Access to register WDT_CON0 was attempted improperly, or
- The Watchdog Timer overflowed either in Time-Out Mode or in Normal Watchdog Timer Mode.

When the CPU is in Idle Mode and the Watchdog Timer is not disabled, an increment of the Watchdog Timer counter from $7FFF_H$ to 8000_H (that is, when bit 15 of the timer is set to 1) sets the NMIWDT bit to wake up the CPU.

16 FPI Bus and Bus Control

This chapter gives an overview on the internal Flexible Peripheral Interconnect (FPI) Bus, and describes the Bus Control Unit (BCU) for the TC1765. Topics covered include the FPI Bus characteristics, BCU, bus arbitration, scheduling, prioritizing, error conditions, and debugging support.

16.1 FPI Bus Overview

The FPI Bus interconnects the functional units of the TC1765, such as the CPU and on-chip peripheral components. The FPI Bus also interconnects the TC1765 to external components by way of the External Bus Controller Unit (EBU). **Figure 16-1** gives an overview of the FPI Bus and the modules connected with it.

The FPI Bus is designed to be quick to acquire by on-chip functional units, and quick to transfer data. The low setup overhead of the FPI Bus access protocol guarantees fast FPI Bus acquisition, which is required for time-critical applications.

The FPI Bus is designed to sustain high transfer rates. For example, a peak transfer rate of up to 160 MBytes/s can be achieved with a 40 MHz bus clock and 32-bit data bus. Multiple data transfers per bus arbitration cycle allow the FPI Bus to operate at close to its peak bandwidth.

Additional features of the FPI Bus include:

- Supports multiple bus masters
- Supports demultiplexed address/data operation
- Address and data buses are 32 bits wide
- Data transfer types include 8-, 16-, and 32-bit sizes
- Single- and multiple-data transfers per bus acquisition cycle
- Designed to minimize EMI and power consumption

Functional units of the TC1765 are connected to the FPI Bus via FPI Bus interfaces. FPI Bus interfaces act as bus agents, requesting bus transactions on behalf of their functional unit, or responding to transaction requests.

There are two types of bus agents:

- Master agents can initiate FPI Bus transaction requests
- Slave agents respond to FPI Bus transaction requests to read or write internal registers and memories

When an FPI Bus master seeks to initiate a transfer on the FPI Bus, it first signals a request for bus ownership. When bus ownership is granted, it initiates an FPI Bus read or write transaction. The unit targeted by the transaction becomes the FPI Bus slave, and responds with the requested action.

Some functional units operate only as slaves, while others can operate as either masters or slaves. The CPU (via the DMU or the PMU) and Direct Memory Access Controller (DMA Sub-Block A0 and DMA Sub-Block A1) typically operate as FPI Bus masters. On-

chip peripheral units are typically FPI Bus slaves. In **Figure 16-1**, the type of interface of the various modules in the TC1765 can be seen (M/S = Master/Slave interface).

FPI Bus arbitration is performed by the on-chip FPI Bus Control Unit. In case of bus errors, the BCU generates an interrupt request to the CPU, and can provide debugging information about the error to the CPU.

For fast external Burst Flash instruction memory operation, the EBU of the TC1765 has a direct path to the PMU. Therefore, external Burst Flash instruction memory accesses can be executed without using the FPI Bus.

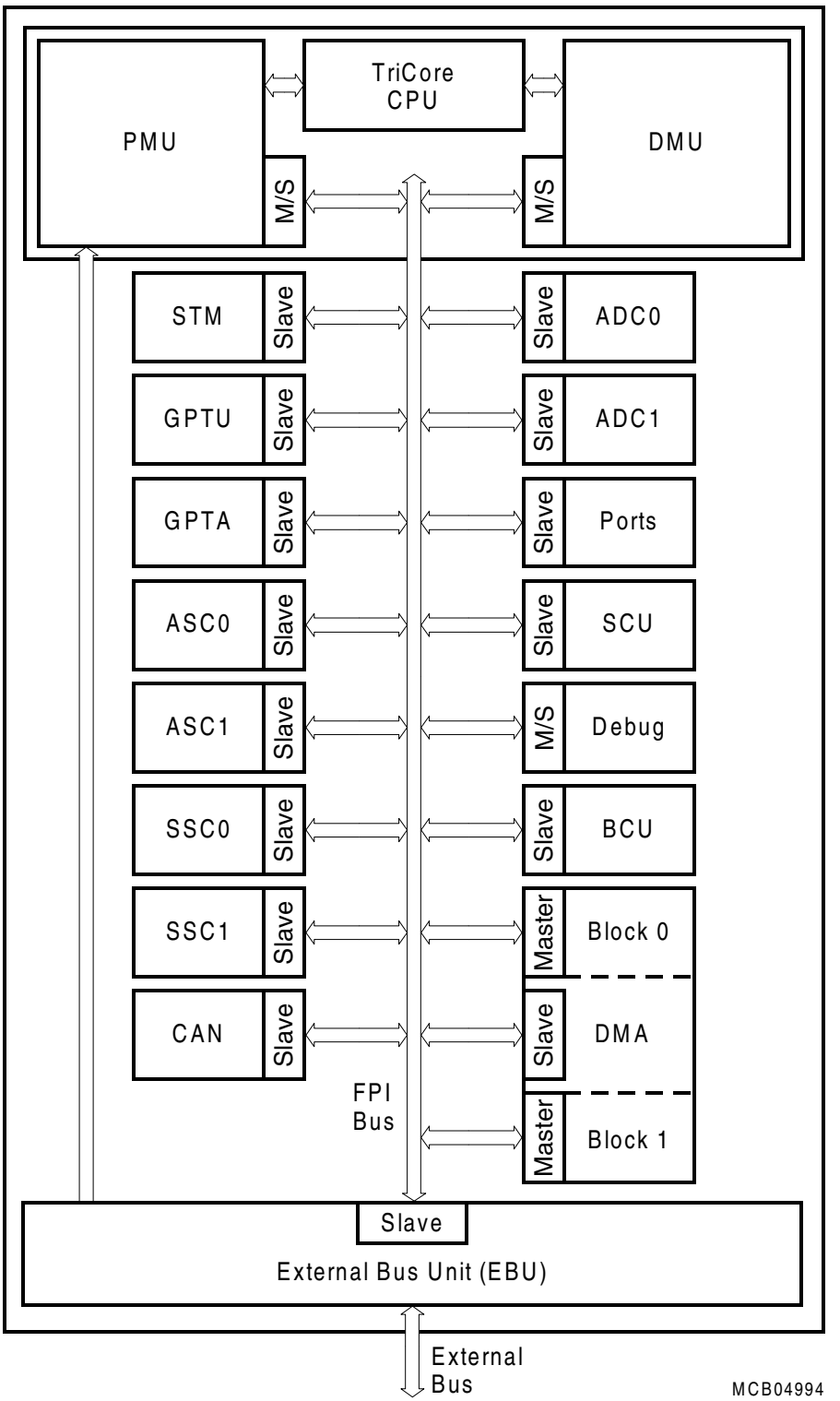


Figure 16-1 TC1765 FPI Bus Block Diagram

16.2 Bus Control Unit

The on-chip FPI Bus Control Unit (BCU) provides bus arbitration, bus error handling, and debug information for error cases. Its design optimizes the speed of bus arbitration. Additionally, it is designed for low power consumption and low EMI.

The BCU arbitrates among the FPI Bus agents to determine the next FPI Bus master. It drives the bus if no other FPI Bus agent is assigned bus ownership to prevent the FPI Bus from electrically floating. It acts as a bus slave when its registers are targeted by an FPI Bus transaction.

Figure 16-2 is a block diagram of the BCU.

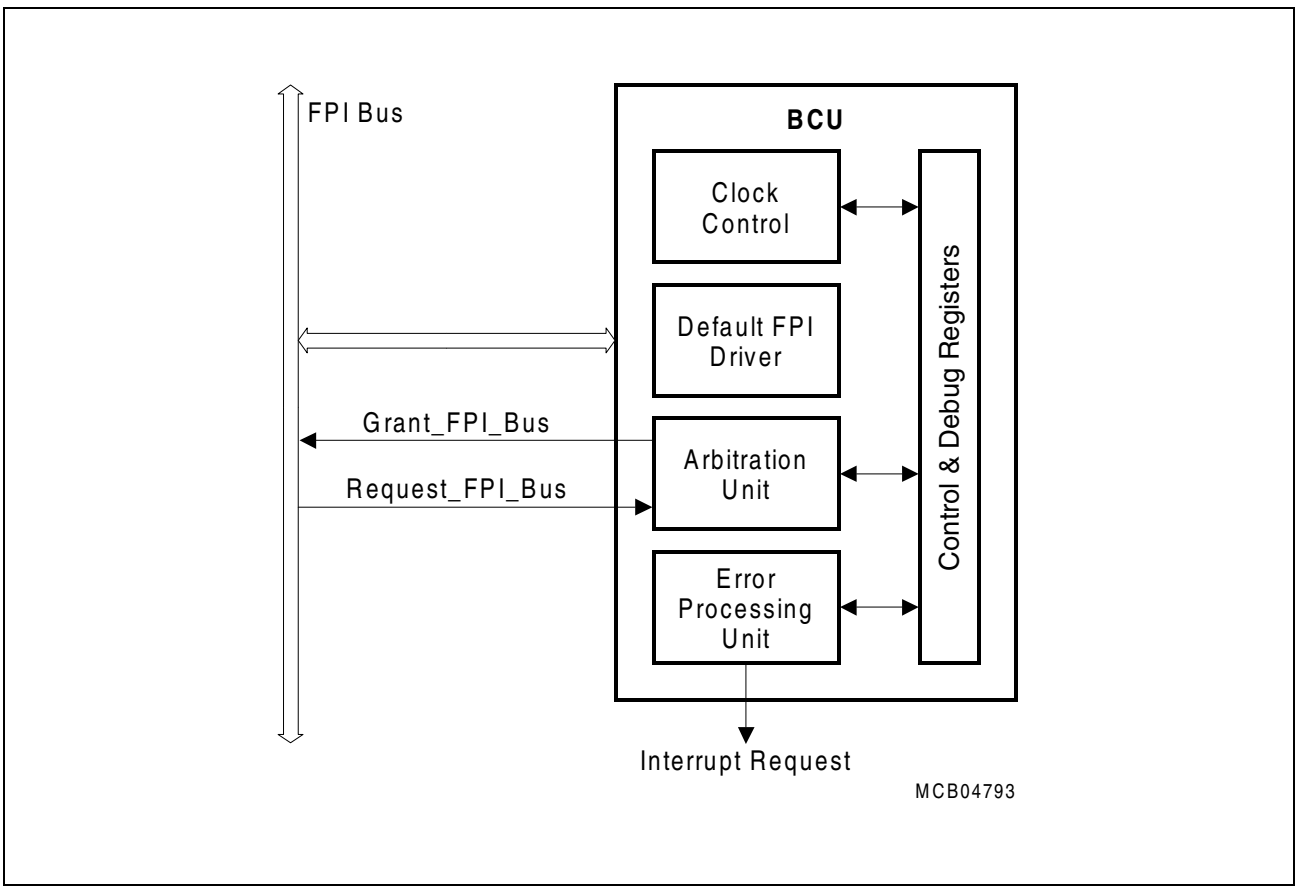


Figure 16-2 FPI Bus Control Unit Block Diagram

The Error Processing Unit is responsible for gathering information and loading the debug registers in the event of a bus error. The default FPI driver becomes active only when no other bus master is able to drive the bus. The clock control unit, if enabled, awakens the BCU only as needed. The “Request_FPI_Bus” lines signal a request to the BCU from a bus master and the “Grant_FPI_Bus” lines are used to grant bus ownership. The control registers control the general operation of the BCU.

16.2.1 FPI Bus Arbitration

The arbitration unit (AB) of the BCU determines whether it is necessary to arbitrate for FPI Bus ownership, and, if so, which available bus requestor gets the FPI Bus for the next data transfer. During arbitration, the bus is granted to the requesting agent with the highest priority. If no request is pending, the bus is granted to a default master. If no bus master takes the bus, the BCU itself will drive the FPI Bus to prevent it from floating electrically.

16.2.1.1 Arbitration Priority

The TC1765 has six bus agents that can become bus master. Each agent is supplied a pre-specified arbitration priority, as shown in **Table 16-1**.

Table 16-1 Priority of TC1765 FPI Bus Agents

Priority	Agent	Comment
highest	Any bus requestor meeting the starvation protection criteria is assigned this priority	Highest priority, used only for starvation protection
	On-Chip Debug System / Cerberus	–
	DMA Controller Sub-Block A1	–
	Data Memory Unit	Default master 1
	Program Memory Unit	Default master 2
	DMA Controller Sub-Block A0	–
lowest	On-Chip Debug System / Cerberus	–

In normal operation, either the DMU or the PMU automatically serves as default master. The bus is granted to this default master which has been at least the default master, whenever there is no request from any other bus master. In this way, the bus is always driven by one of the masters. In some exceptional circumstances, however, the BCU must drive the FPI Bus. These conditions include:

- After reset
- A non-existing module is accessed (error)
- A time-out condition occurs (error)
- No other master can be granted the FPI Bus because of special conditions

16.2.1.2 Bus Starvation Protection

Because assignment of priorities to these six bus agents is fixed, it is possible that a lower-priority requestor may never be granted the bus if a higher-priority requestor continuously asks for, and receives, bus ownership. To protect against bus starvation of lower-priority masters, an optional feature of the TC1765 will detect such cases and momentarily raise the priority of the lower-priority requestor to the highest priority (above all other priorities), thereby guaranteeing it access.

Starvation protection employs a counter which is incremented each time an arbitration is performed by the BCU. When this counter reaches a user-programmable threshold value, all the bus request lines are sampled, and for each active bus request, a request flag is set in an internal BCU register. This flag is reset automatically when a master is granted the bus.

When the counter reaches the threshold value, it is automatically reset to zero and starts counting up again. When the next period is finished, the request lines are sampled again. If an active request is detected, for which the request flag set during the last sample is still set, this means that this master was not granted the bus during the previous period. This master will now be set to the highest priority and will be granted service. If there are several masters for which this starvation condition applies, they are served in the order of their hardwired priority ranking.

Starvation protection can be enabled and disabled through the BCU_CON.SPE bit. The sample period of the counter is programmed through the BCU_CON.SPC bit field. This bit field should be set to a value at least greater than or equal to the number of masters. Its reset value is 40_H.

16.2.2 Error Handling

Two classes of error condition can arise on the FPI Bus:

1. A slave indicates a severe problem such as an unaligned data access request, by returning an error code instead of an acknowledge.
2. A time-out is detected for the current bus operation, indicating a non-responding slave.

A bus error condition causes the BCU to issue an interrupt request to the CPU, and if enabled, causes the BCU to capture information about the bus error condition for debugging.

Bus error information gathering is enabled by default. It can be disabled by setting bit CON.DBG to 0. If a bus error occurs when enabled, the status of the bus, including address, data, and the control information, is captured into registers BCU_EADD, BCU_EDAT and BCU_ECON, respectively. Kernel software must read the debug information in response to the interrupt to examine and resolve the problem.

Note: If the CPU itself caused the bus error either through a load/store operation via the DMU or an instruction fetch operation via the PMU, a bus trap is issued to the CPU in addition to the interrupt issued by the BCU. To handle this condition, the trap routine in the kernel software must read the BCU error status registers and then clear the interrupt request from the BCU.

Interpreting the BCU Error Information

Some knowledge about the operation of the internal FPI Bus is required in order to interpret the captured information in case of a bus error. Although the captured address and data values captured in registers BCU_EADD and BCU_EDAT, respectively, are self-explanatory, the captured FPI Bus control information needs some more explanation.

Register BCU_ECON captures the state of the read (RDN), write (WRN), Supervisor Mode (SVM), acknowledge (ACK), ready (RDY), abort (ABT), time-out (TOUT), identification (TAG) and operation code (OPC) lines of the FPI Bus.

The read and write signals are active-low. For regular read or write accesses, only one of these lines is activated (set to 0). There is one special case defined for the FPI Bus. If a master performs a read-modify-write transaction (for example, to modify a bit in a peripheral register), this transaction is indicated by both lines, read and write, being activated in the first access (read access).

The supervisor mode signal is set to 1 for an access in Supervisor Mode, and set to 0 for an access in User Mode.

The ready signal indicates the end of a transfer. It is normally driven to 1 in the (last) data cycle. During wait state insertion, ready is driven to 0.

Under certain conditions, a master can abort a transfer that has already started. This is indicated with the abort signal set to 0.

The time-out signal indicates if there was no response on the bus to an access, and the programmed time (via BCU_CON.TOUT) has elapsed. TOUT is set to one in this case.

An acknowledge code has to be driven by the selected slave during each data cycle of an access.These codes are listed in **Table 16-2**.

Table 16-2 FPI Bus Acknowledge Codes

Code (ACK)	Description
00_B	NSC: No Special Condition.
01_B	SPT: Split Transaction (not used in the TC1765).
10_B	RTY: Retry. Slave can currently not respond to the access. Master needs to repeat the access later.
11_B	ERR: Bus Error, last data cycle is aborted.

Each master on the FPI Bus is assigned a 4-bit identification number, the TAG (see Table 16-3). This allows to distinguish which master has performed the current transaction.

Table 16-3 FPI Bus TAG Assignments in the TC1765

TAG-Number	Module	Description
0	TCU	Test Control Unit
1	–	Reserved
2	–	Reserved
3	DMU	Data Memory Unit
4	PMU	Program Memory Unit
5	DMA	DMA Controller Sub Block A0
6		DMA Controller Sub Block A1
7 … 15	–	Reserved

Transactions on the FPI Bus are classified via a 4-bit operation code, listed in Table 16-4. Note that block transfers (OPC = 0100_B to 0110_B) and split transactions (OPC = 1000_B to 1110_B) are not used in the TC1765.

Table 16-4 FPI Bus Operation Codes (OPC)

OPC	Description	OPC	Description
0000_B	Single Byte Transfer (8-bit)	**1000_B**	Split Block Transfer Request (1 transfer)
0001_B	Single Half-Word Transfer (16-bit)	**1001_B**	Split Block Transfer Request (2 transfers)
0010_B	Single Word Transfer (32-bit)	**1010_B**	Split Block Transfer Request (4 transfers)
0011_B	Single Double-Word Transfer (64-bit)	**1011_B**	Split Block Transfer Request (8 transfers)
0100_B	2-Word Block Transfer	**1100_B**	Split Block Response
0101_B	4-Word Block Transfer	**1101_B**	Split Block Failure
0110_B	8-Word Block Transfer	**1110_B**	Split Block End
0111_B	Reserved	**1111**	No operation

Note: Shaded FPI Bus transactions are not used in the TC1765.

16.2.3 BCU Power Saving Mode

The BCU can be configured so that it shuts down automatically when not needed by disabling its internal clock. When it is needed again, for instance when a bus request signal is received from a master, the BCU will enable its clock and perform the arbitration. If no further bus activity is required after the transfer has completed, the BCU will automatically shut off its clock and return to idle mode.

Automatic power management is controlled through the BCU_CON.PSE bit. When cleared to 0, power management is disabled, and the BCU clock is always active. This might be required, for instance, to debug both the active and idle FPI Bus states of an application via an external emulator or other debugging hardware.

16.2.4 BCU Registers

The five BCU registers can be divided into three types, as shown in **Figure 16-3**.

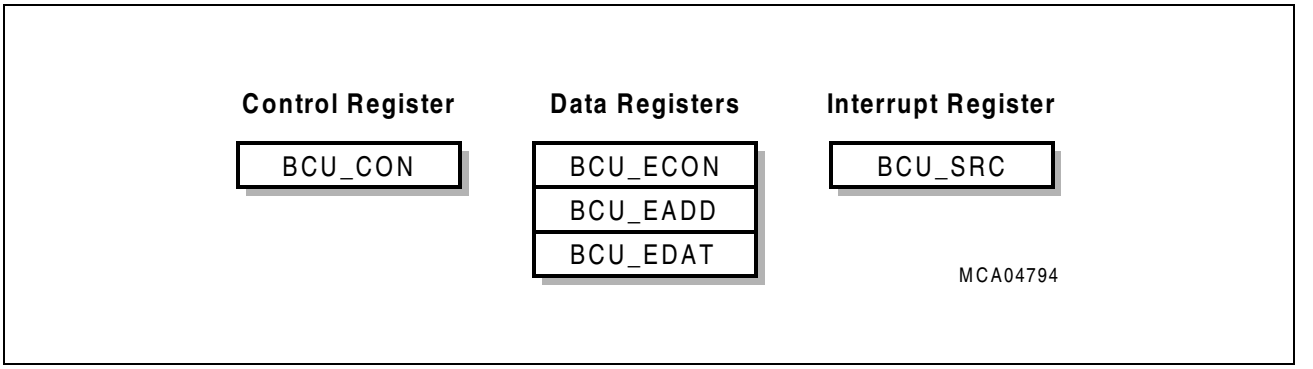


Figure 16-3 BCU Registers

Table 16-5 BCU Registers

Register Short Name	Register Long Name	Offset Address	Description see
BCU_CON	BCU Control Register	0010_H	**Page 16-11**
BCU_ECON	BCU Error Control Capture Register	0020_H	**Page 16-13**
BCU_EADD	BCU Error Address Capture Register	0024_H	**Page 16-14**
BCU_EDAT	BCU Error Data Capture Register	0028_H	**Page 16-14**
BCU_SRC	BCU Service Request Control Register	$00FC_H$	**Page 16-15**

In the TC1765, the registers of the BCU are located in the address range:

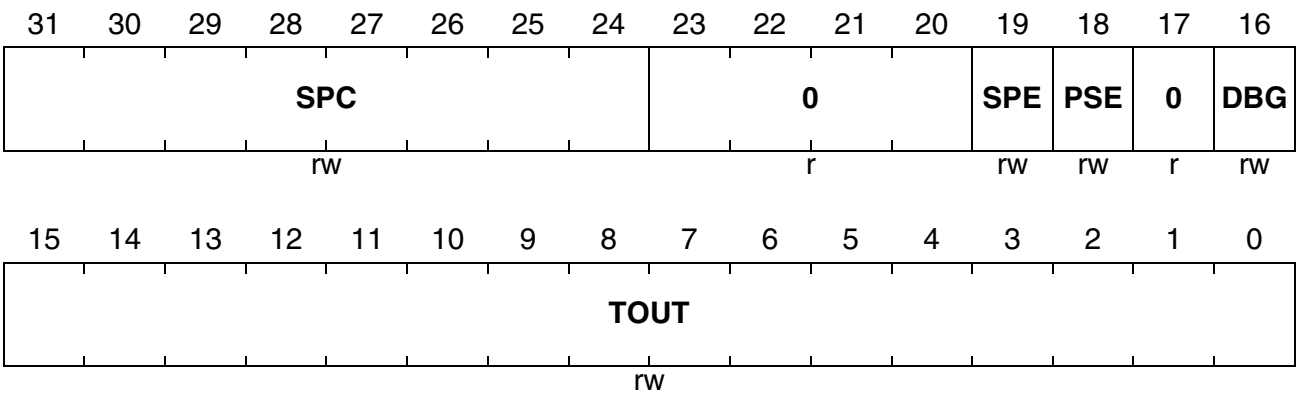
- Module Base Address: F000 0200_H
  Module End Address: F000 $02FF_H$
- Absolute Register Address = Module Base Address + Offset Address (see **Table 16-5**)

Note: The BCU allows word accesses only (32-bit) to its control and data registers. Byte and half-word accesses will result in a bus error.

16.2.4.1 BCU Control Register

The BCU Control Register controls the overall operation of the BCU, including setting the starvation sample period, the bus time-out period, enabling starvation-protection mode, and error handling.

BCU_CON
BCU Control Register **Reset Value: 4009 FFFF$_H$**



Field	Bits	Type	Description
TOUT	[15:0]	rw	**BCU Bus Time-Out Value** The bit field defines the number of FPI Bus time-out cycles. Default after reset is FFFF$_H$ (= 65536 bus cycles).
DBG	16	rw	**BCU Debug Trace Enable** 0 BCU debug trace disabled. No error information captured. 1 BCU debug trace enabled. Error information is captured in registers BCU_EADD, BCU_EDAT, and BCU_ECON (default after reset).
PSE	18	rw	**BCU Power Saving (Automatic Clock Control) Enable** 0 BCU power saving disabled (default after reset) 1 BCU power saving enabled
SPE	19	rw	**BCU Starvation Protection Enable** 0 BCU protection disabled 1 BCU protection enabled (default after reset)

Field	Bits	Type	Description
SPC	[31:24]	rw	**BCU Sample Period Control** Defines the sample period for the starvation counter. Must be larger than the number of masters. The reset value is 40_H.
0	17, [23:20]	r	**Reserved**; read as 0; should be written with 0.

16.2.4.2 BCU Debug Registers

The capture of bus error conditions is enabled by setting BCU_CON.DBG to 1. In case of a bus error, information about the condition will then be stored in the BCU debug registers. The BCU debug registers can then be examined by software to help determine the cause of the error.

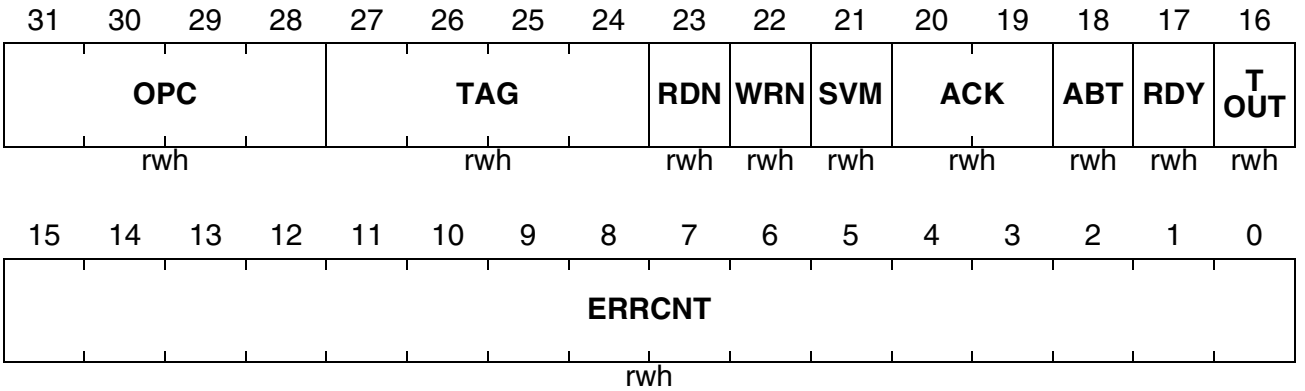
If enabled, and a bus error occurs, the BCU Error Control Capture Register, BCU_ECON, will hold the captured FPI Bus control information, and a count of the number of bus errors. The BCU Error Address Capture Register, BCU_EADD, will store the captured FPI Bus address, and the BCU Error Data Capture Register, BCU_EDAT, will store the captured FPI Bus data.

If the capture of bus error conditions is disabled (BCU_CON.DBG = 0), these registers remain untouched.

Note: These registers store only for the first error. In case of multiple bus errors, an error counter BCU_ECON[15:0] shows the number of bus errors since the first error occurred. A hardware reset clears this 16-bit counter to zero, but the counter can be set to any value through software. This counter is prevented from overflowing, so a value of 2^{16} - 1 indicates that at least this many errors have occurred, but there may have been more. After BCU_ECON has been read, the BCU_ECON, BCU_EADD and BCU_EDAT registers are re-enabled to trace FPI Bus activity.

BCU_ECON
BCU Error Control Capture Register

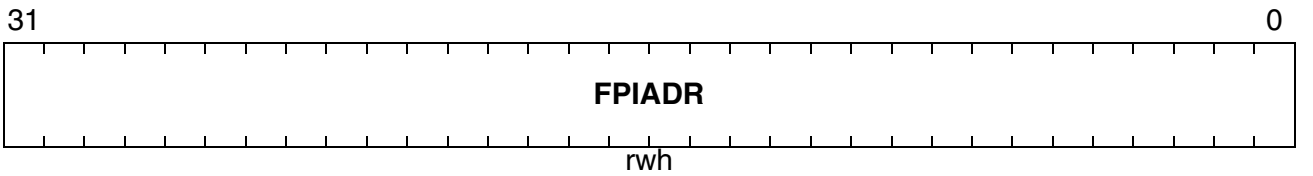
Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
ERRCNT	[15:0]	rwh	**Number of FPI Bus Error Counter** ERRCNT is incremented on each occurrence of an FPI Bus error. ERRCNT is reset to 0000$_H$ after the BCU_ECON register is read.
TOUT	16	rwh	**State of FPI Bus Time-Out Signal (active high)**
RDY	17	rwh	**State of FPI Bus Ready Signal** (active high)
ABT	18	rwh	**State of FPI Bus Abort Signal** (active low)
ACK	[20:19]	rwh	**State of FPI Bus Acknowledge Signal**
SVM	21	rwh	**State of FPI Bus Supervisor Mode Signal** (active high)
WRN	22	rwh	**State of FPI Bus Write Signal** (active low)
RDN	23	rwh	**State of FPI Bus Read Signal** (active low)
TAG	[27:24]	rwh	**FPI Bus Tag Number** see **Table 16-3**
OPC	[31:28]	rwh	**FPI Bus Operation Code** see **Table 16-4**

BCU_EADD
BCU Error Address Capture Register

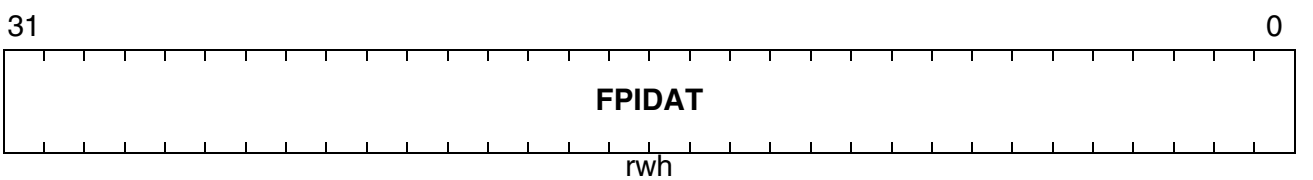
Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
FPIADR	[31:0]	rwh	**Captured FPI Bus Address (in case of a bus error)** Note: If there are multiple errors, only the address of the first error is captured.

BCU_EDAT
BCU Error Data Capture Register

Reset Value: 0000 0000$_H$

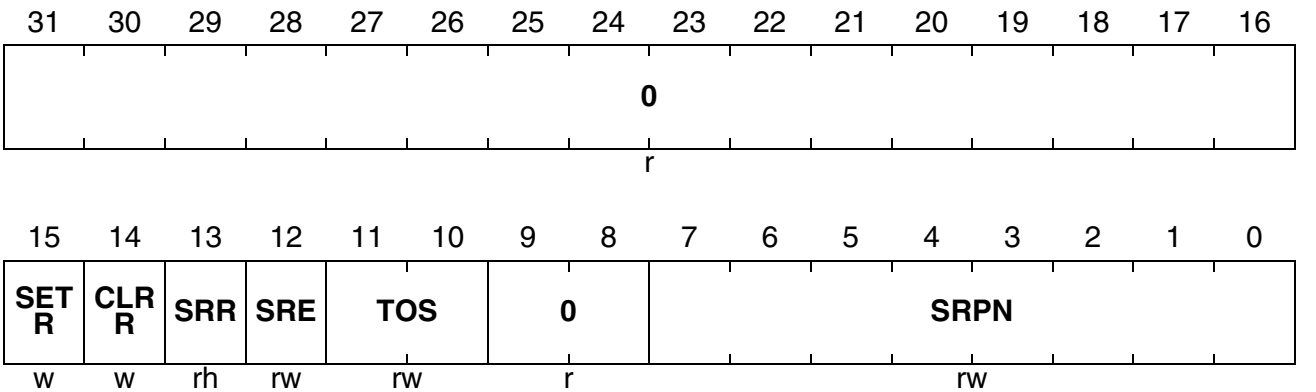


Field	Bits	Type	Description
FPIDAT	[31:0]	rwh	**Captured FPI Bus Data (in case of a bus error)** Note: If there are multiple errors, only the data for the first error are captured.

16.2.4.3 BCU Service Request Control Register

In case of a bus error, the BCU generates an interrupt request to the selected service provider (usually the CPU). This interrupt request is controlled through a standard service request control register.

BCU_SRC
BCU Service Request Control Register **Reset Values: 0000 0000$_H$**



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	[11:10]	rw	**Type of Service Control**
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], [31:16]	r	**Reserved**; read as 0; should be written with 0.

*Note: Further details on interrupt handling and processing are described in **Chapter 14** in this User's Manual.*

17 System Timer

17.1 Overview

This chapter describes the System Timer (STM). The TC1765's STM is designed for global system timing applications requiring both high precision and long range. The STM has the following features:

- Free-running 56-bit counter
- All 56 bits can be read synchronously
- Different 32-bit portions of the 56-bit counter can be read synchronously
- Driven by clock, f_{STM} (identical to the system clock f_{SYS}).
- Counting starts automatically after a reset operation
- STM is reset under following reset causes:
  - Wake-up reset (PMG_CON.DSRW must be set)
  - Watchdog reset
  - Software reset (RST_REQ.RRSTM must be set)
  - Power-on reset
- STM is not reset at a hardware reset ($\overline{HDRST}$ = 0)

Special STM register semantics provide synchronous views of the entire 56-bit counter, or 32-bit subsets at different levels of resolution.

The maximum clock period is $2^{56} \times 1 / f_{STM}$. At f_{STM} = 40 MHz, for example, the STM counts 57.1 years before overflowing. Thus, it is capable of continuously timing the entire expected product life-time of a system without overflowing.

17.2 Kernel Functions

The STM is an upward counter, running with the system clock frequency ($f_{STM} = f_{SYS}$). It is enabled per default after reset, and immediately starts counting up. Other than via reset, it is no possible to affect the contents of the timer during normal operation of the application, it can only be read, but not written to. Depending on the implementation of the clock control of the STM, the timer can optionally be disabled or suspended for power-saving and debugging purposes via a clock control register.

Due to the 56-bit width of the STM, it is not possible to read its entire contents with one instruction. It needs to be read with two load instructions. Since the timer would continue to count between the two load operations, there is a chance that the two values read are not be consistent (due to possible overflow from the low part of the timer to the high part between the two read operations). To enable a synchronous and consistent reading of the STM contents, a capture register (CAP), is implemented. It latches the contents of the high part of the STM each time the low part, TIM0, is read. Thus, it holds the upper value of the timer at exactly the same time when the lower part is read. The second read operation would then read the contents of the CAP to get the complete timer value.

The System Timer can also be read in sections from seven registers, TIM0 through TIM6, which select increasingly higher-order 32-bit ranges of the System Timer. These can be viewed as individual 32-bit timers, each with a different resolution and timing range.

Figure 17-1 is an overview on the System Timer module. It shows the options for reading parts of STM contents.

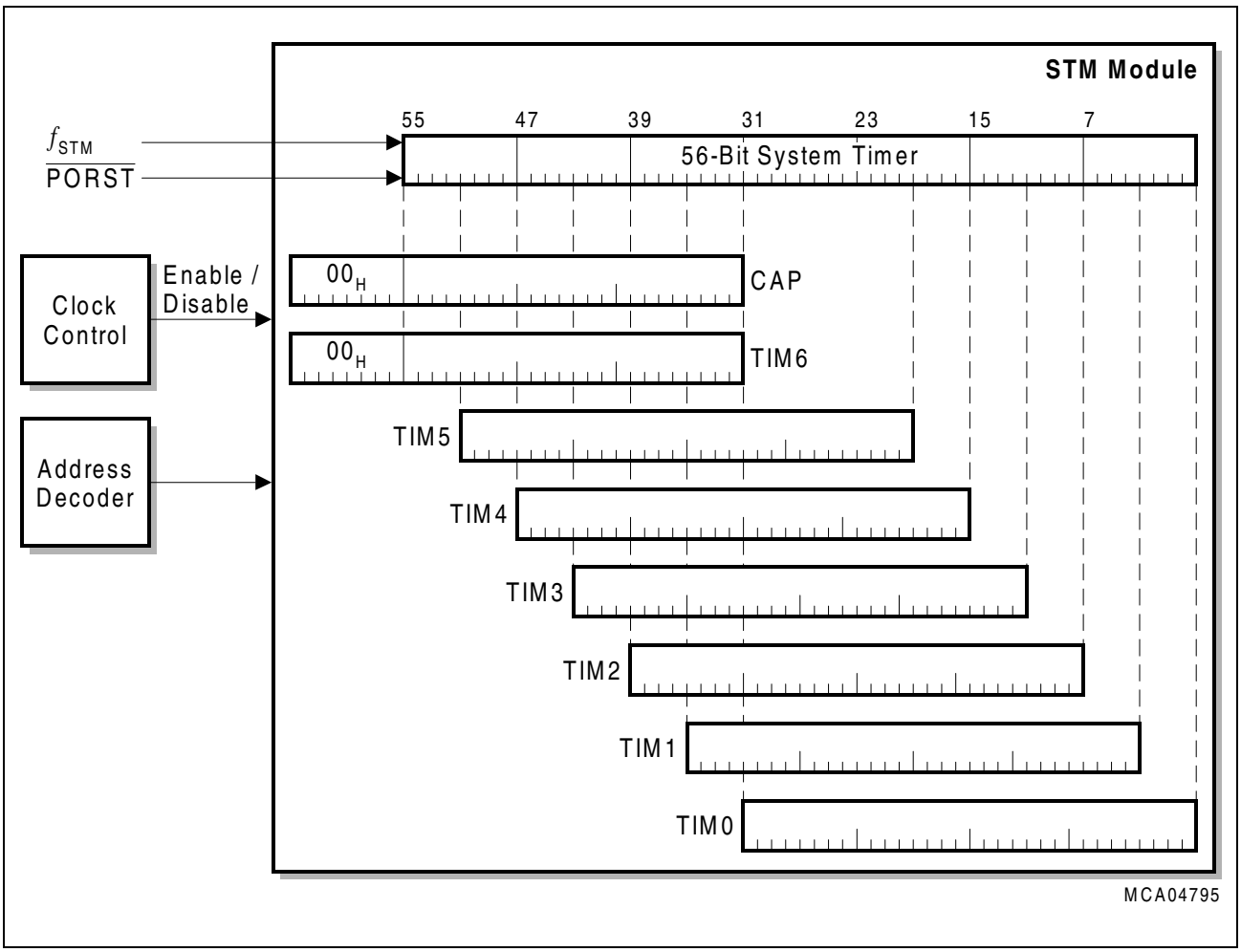


Figure 17-1 General Block Diagram of the STM Module

Table 17-1 is an overview on the individual timer registers with their resolution and timing range. As an example, the values for a 40 MHz system frequency are given.

Table 17-1 System Timer Resolutions and Ranges

Register	STM Bits	Resolution [s]	Range [s]	Example Frequency: 40 MHz $f_{STM} = f_{SYS}$	
				Resolution	Range
TIM0	[31:0]	f_{STM}	$2^{32} / f_{STM}$	25 ns	107.4 s
TIM1	[35:4]	$16 / f_{STM}$	$2^{36} / f_{STM}$	400 ns	1717.9 s
TIM2	[39:8]	$256 / f_{STM}$	$2^{40} / f_{STM}$	6.4 µs	458.1 min
TIM3	[43:12]	$4096 / f_{STM}$	$2^{44} / f_{STM}$	102.4 µs	122.2 h
TIM4	[47:16]	$65536 / f_{STM}$	$2^{48} / f_{STM}$	1.64 ms	81.45 days
TIM5	[51:20]	$2^{20} / f_{STM}$	$2^{52} / f_{STM}$	26.2 ms	3.57 yr
TIM6	[55:32]	$2^{32} / f_{STM}$	$2^{56} / f_{STM}$	107.4 s	57.1 yr
CAP	[55:32]	$2^{32} / f_{STM}$	$2^{56} / f_{STM}$	107.4 s	57.1 yr

17.3 Kernel Registers

The STM registers can be divided into two types, as shown in Figure 17-2.

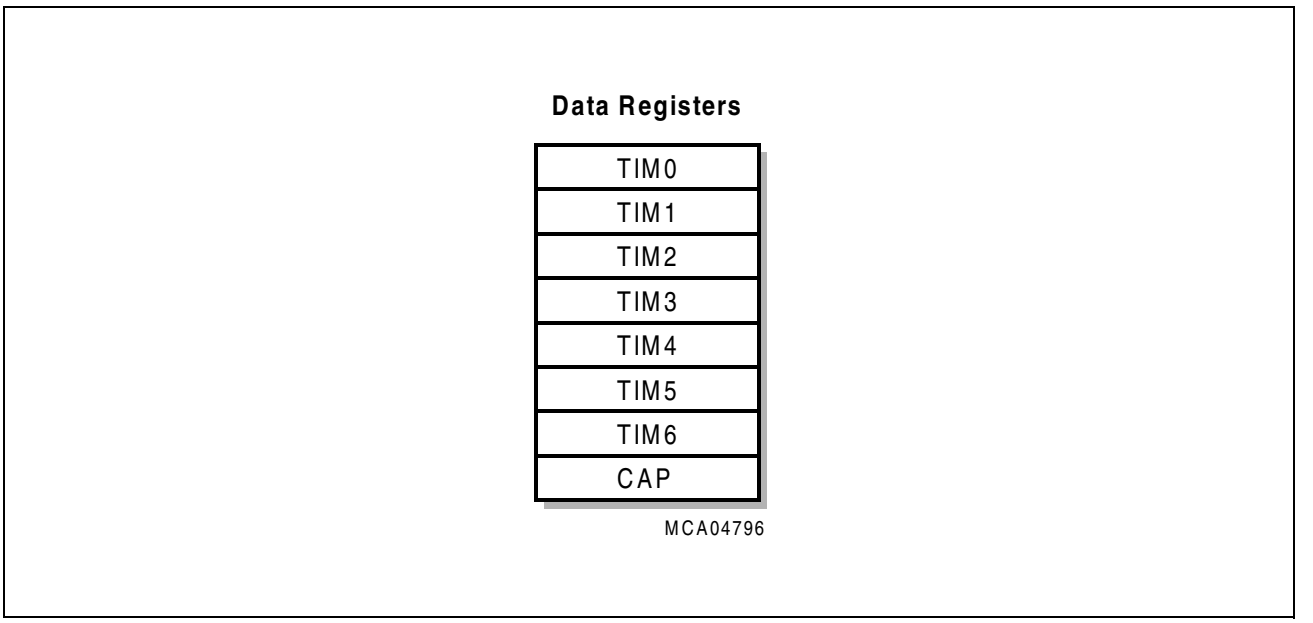


Figure 17-2 SFRs of the STM Module

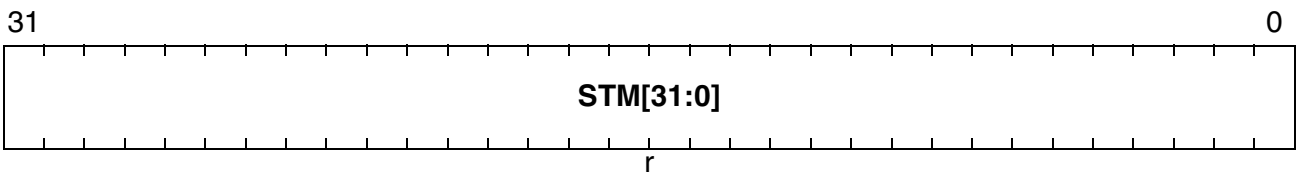
Table 17-2 STM Kernel Registers

Register Short Name	Register Long Name	Offset Address	Description see
TIM0	Timer Register 0	0010_H	Page 17-5
TIM1	Timer Register 1	0014_H	Page 17-5
TIM2	Timer Register 2	0018_H	Page 17-5
TIM3	Timer Register 3	$001C_H$	Page 17-5
TIM4	Timer Register 4	0020_H	Page 17-6
TIM5	Timer Register 5	0024_H	Page 17-6
TIM6	Timer Register 6	0028_H	Page 17-6
CAP	Timer Capture Register	$002C_H$	Page 17-6

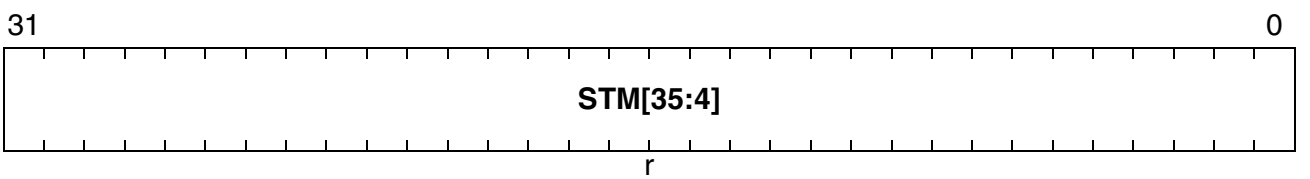
Note: All STM kernel register names described in this section will be referenced in other parts of this TC1765 User's Manual with the module name prefix "STM_".

TIM1 to TIM6 provide 32-bit views at varying resolutions of the underlying STM counter.

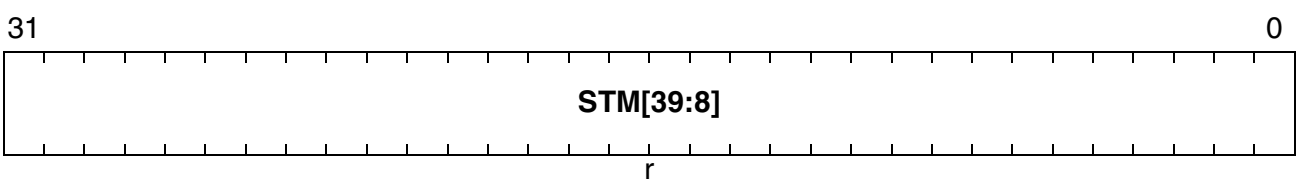
TIM0
Timer Register 0 **Reset Value: 0000 0000$_H$**



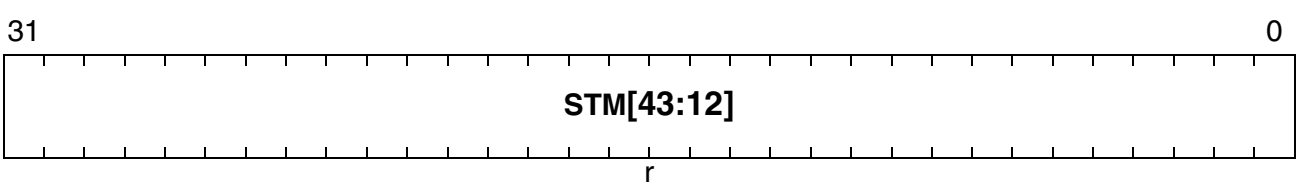
TIM1
Timer Register 1 **Reset Value: 0000 0000$_H$**



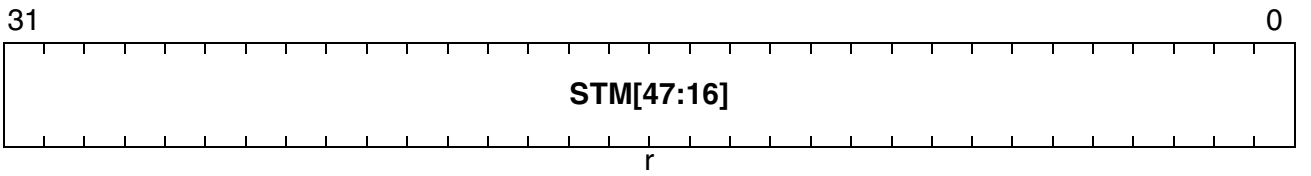
TIM2
Timer Register 2 **Reset Value: 0000 0000$_H$**



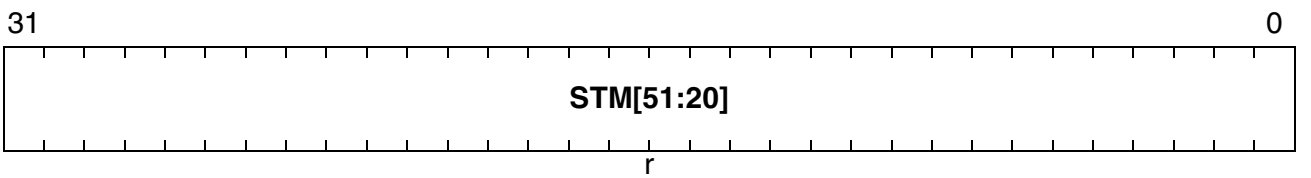
TIM3
Timer Register 3 **Reset Value: 0000 0000$_H$**



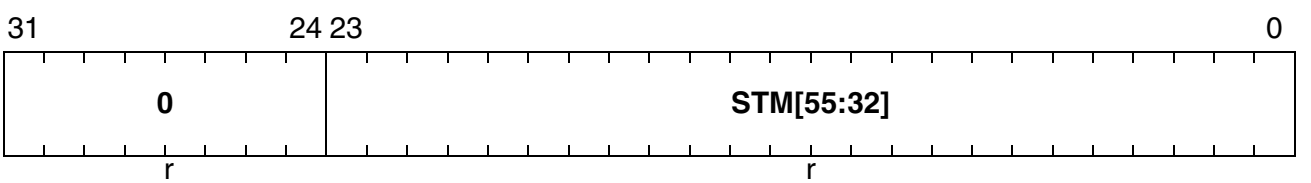
TIM4
Timer Register 4 **Reset Value: 0000 0000$_H$**



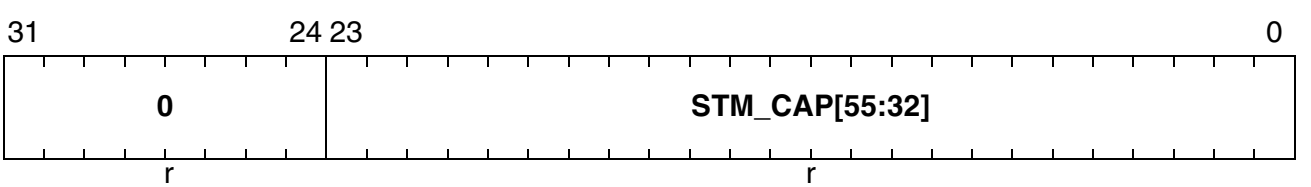
TIM5
Timer Register 5 **Reset Value: 0000 0000$_H$**



TIM6
Timer Register 6 **Reset Value: 0000 0000$_H$**



CAP
Timer Capture Register **Reset Value: 0000 0000$_H$**



Note: CAP captures the system timer bits [55:32] when a read of TIM0 (contains the system timer bits [31:0]) is performed in order to enable software to operate with a coherent value of all the 56 bits of the system timer.

Note: The bits in registers CAP - TIM0 are all read only.

17.4 External Register

The clock control register allows to switch the System Timer on or off. After power-on reset, the System Timer is always enabled and starts counting. The System Timer can be disabled by setting bit DISR to 1.

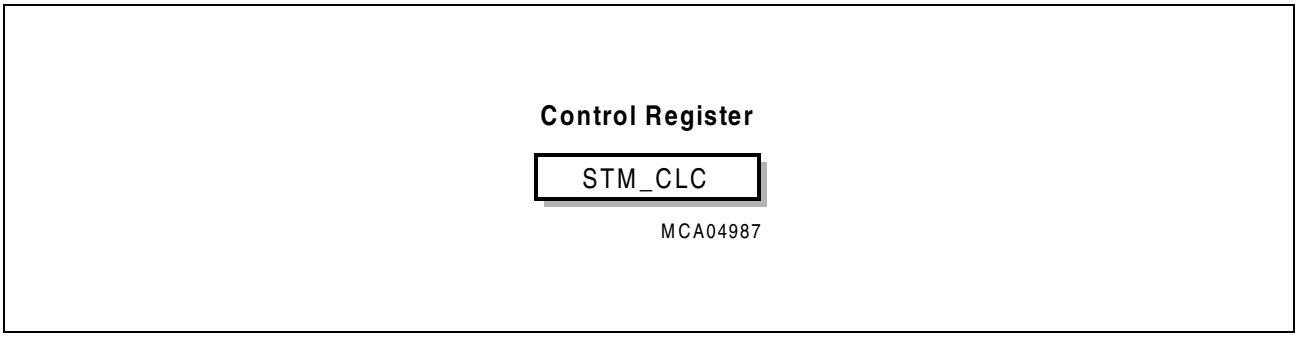


Figure 17-3 STM External Register

STM_CLC
System Timer Clock Control Register **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0														DIS S	DIS R
r														r	rw

Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module. 0 No disable requested 1 Disable requested
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module 0 Module is enabled 1 Module is disabled
0	[31:2]	r	**Reserved**; read as 0; should be written with 0.

17.5 STM Register Address Ranges

In the TC1765, the registers of the STM module are located in the following address range:

- Module Base Address: F000 0300_H
  Module End Address: F000 $03FF_H$
- Absolute Register Address = Module Base Address + Offset Address
  (offset addresses see **Table 17-2**)

18 Watchdog Timer

This chapter describes the TC1765 Watchdog Timer (WDT). Topics include an overview of the Watchdog Timer function and descriptions of the registers, the password protection scheme, accessing registers, modes, and initialization.

18.1 Watchdog Timer Overview

The Watchdog Timer (WDT) provides a highly reliable and secure way to detect and recover from software or hardware failure. The WDT helps to abort an accidental malfunction of the TC1765 in a user-specified time period. When enabled, the WDT will cause the TC1765 system to be reset if the WDT is not serviced within a user-programmable time period. The CPU must service the WDT within this time interval to prevent the WDT from causing a TC1765 system reset. Hence, routine service of the WDT confirms that the system is functioning properly.

In addition to this standard “Watchdog” function, the WDT incorporates the EndInit feature and monitors its modifications. A system-wide line is connected to the ENDINIT bit implemented in a WDT control register, serving as an additional write-protection for critical registers (besides Supervisor Mode protection). Registers protected via this line can only be modified when Supervisor Mode is active and bit ENDINIT = 0.

Because servicing the Watchdog and modifications of the ENDINIT bit are critical functions that must not be allowed in case of a system malfunction, a sophisticated scheme is implemented which requires a password and guard bits during accesses to the WDT control register. Any write access that does not deliver the correct password or the correct value for the guard bits is regarded as a malfunction of the system, and a Watchdog reset is triggered. In addition, even after a valid access has been performed and the ENDINIT bit has been cleared to provide access to the critical registers, the Watchdog imposes a time-limit for this access window. If ENDINIT has not been properly set again before this limit expires, the system is assumed to malfunction, and a Watchdog reset is triggered. These stringent requirements, although not a guarantee, nonetheless provide a high degree of assurance of the robustness of system operation.

A further enhancement in the TC1765’s Watchdog Timer is its reset prewarning operation. Instead of immediately resetting the device on the detection of an error, as known from standard Watchdogs, the WDT first issues an Non-maskable Interrupt (NMI) to the CPU before finally resetting the device at a specified time period later. This gives the CPU a chance to save system state to memory for later examination of the cause of the malfunction, an important aid in debugging.

18.2 Features of the Watchdog Timer

The major features of the WDT are summarized here. The Watchdog Timer is implemented in the System Control Unit (SCU) module of the TC1765. **Figure 18-1** gives an overview of its interface signals.

- 16-bit Watchdog counter.
- Selectable input frequency: $f_{SYS}/256$ or $f_{SYS}/16384$.
- 16-bit user-definable reload value for normal Watchdog operation, fixed reload value for Time-Out and Prewarning Modes.
- Incorporation of the ENDINIT bit and monitoring of its modifications.
- Sophisticated password access mechanism with fixed and user-definable password fields.
- Proper access always requires two write accesses. The time between the two accesses is monitored by the WDT and limited.
- Access Error Detection: Invalid password (during first access) or invalid guard bits (during second access) trigger the Watchdog reset generation.
- Overflow Error Detection: An overflow of the counter triggers the Watchdog reset generation.
- Watchdog function can be disabled; access protection and ENDINIT monitor function remain enabled.
- Double Reset Detection: If a Watchdog induced reset occurs twice, a severe system malfunction is assumed and the TC1765 is held in reset until a power-on reset or hardware reset occurs. This prevents the device from being periodically reset if, for instance, connection to the external memory has been lost such that even system initialization could not be performed.
- Important debugging support is provided through the reset prewarning operation by first issuing an NMI to the CPU before finally resetting the device after a certain period of time.

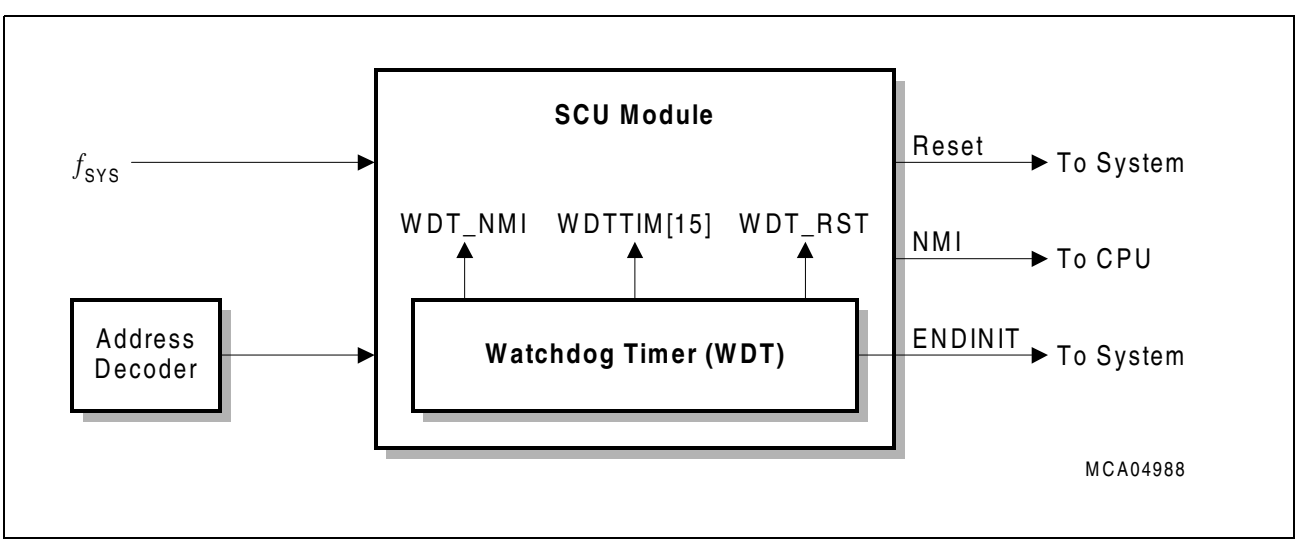


Figure 18-1 Interface of the WDT Inside and Outside the SCU Module

18.3 The EndInit Function

Because understanding of the ENDINIT bit and its function is an important prerequisite for the descriptions in the following sections, its function is explained first.

There are a number of registers in the TC1765 that are usually programmed only once during the initialization sequence of the application. Modification of such registers during normal application run can have a severe impact on the overall operation of modules or the entire system.

While the Supervisor Mode, which allows writes to registers only when it is active, provides a certain level of protection against unintentional modifications, this might not provide enough security for system critical registers.

The TC1765 provides one more level of protection for such registers via the EndInit feature. This is a highly secure write protection scheme that makes unintentional modifications of registers protected by this feature nearly impossible.

The EndInit feature consists of an ENDINIT bit incorporated in the Watchdog Timer control register, WDT_CON0. A system-wide line is connected to this bit. Registers protected via EndInit use the state of this line to determine whether or not writes are enabled. Writes are only enabled if ENDINIT = 0 and Supervisor Mode is active. Write attempts if this condition is not true will cause a bus error, the register contents will not be modified in this case.

An additional line, controlled through a separate bit, to protect against unintentional writes does provide an extra level of security. However, to get the highest level of security, this bit is incorporated in the highly secure access protection scheme implemented in the Watchdog Timer. This is a complex procedure, that makes it nearly impossible for the ENDINIT bit to be modified unintentionally. It is explained in the following sections. In addition, the WDT monitors ENDINIT modifications by starting a time-out sequence each time software opens access to the critical registers through clearing ENDINIT to 0. If the Time-out period ends before ENDINIT is set to 1 again, a malfunction of the software and/or the hardware is assumed and the device is reset.

The access protection scheme and the EndInit time-out operation of the WDT is described in the following sections. **Table 18-1** lists the registers that are protected via the EndInit feature in the TC1765.

Table 18-1 TC1765 Registers Protected via the EndInit Feature

Normal Mode	Description
mod_CLC	All clock control registers of the individual peripheral modules are EndInit-protected.
BTV, BIV, ISP	Trap and interrupt vector table pointer as well as the interrupt stack pointer are EndInit-protected.
WDT_CON1	The Watchdog Timer Control Register 1, which controls the disabling and the input frequency of the Watchdog Timer, is EndInit-protected. In addition, its bits will only have an effect on the WDT when ENDINIT is properly set to 1 again.

18.4 Watchdog Timer Operation

The following sections describe the registers, the operation, and different modes of the WDT, as well as the password access mechanism. **Figure 18-2** gives an example for the operation of the Watchdog Timer. A rough description of the sequence of events in this figure is provided here. Refer to the following sections for a detailed explanation.

1. Time-Out Mode is automatically entered after reset. Timer counts with slowest input clock.
2. Time-Out Mode terminated and Normal Mode is entered by setting ENDINIT to 1.
3. Normal Mode is terminated and Time-Out Mode is entered through a password access to WDT_CON0. The reload value was set to REL_1.
4. Time-Out Mode is terminated and Normal Mode entered again by setting ENDINIT to 1. The reload value WDTREL has been changed to REL_2 and the timer input clock was set to the fast clock.
   Events 3) and 4) constitute a Watchdog Timer service sequence.
5. The Watchdog Timer was not serviced and continued to count until overflow. Reset Prewarning Mode is entered. Timer counts with selected fast input clock. Watchdog operation cannot be altered or stopped in this mode.
6. Timer continued to count until overflow, generating a Watchdog Timer reset.
7. Time-Out Mode is automatically entered after reset. Timer counts with slowest input clock.
8. Time-Out Mode is terminated and Normal Mode is entered again.

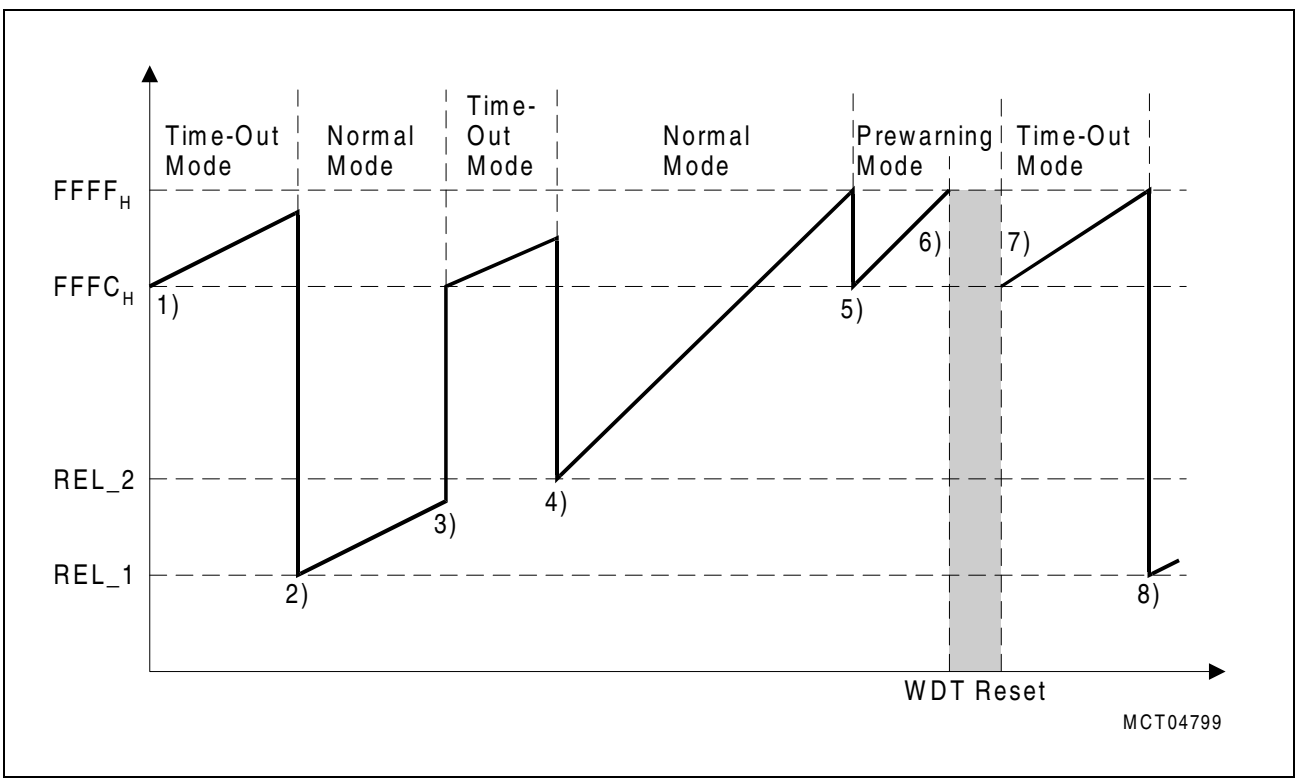


Figure 18-2 Example for an Operation Sequence of the Watchdog Timer

18.4.1 WDT Register Overview

Two control registers, WDT_CON0 and WDT_CON1, and one status register, WDT_SR, serve for communication of the software with the WDT. This section provides a short overview and describes the access mechanisms of the WDT registers. Detailed layout and bit descriptions of the registers are given in **Section 18.6**.

Register WDT_CON0 holds the ENDINIT bit, a register lock status bit (WDTLCK), an 8-bit user-definable password field (WDTPW), and the user-definable reload (start) value (WDTREL) for the Watchdog Timer in Normal Mode.

Register WDT_CON1 contains two bits. Bit WDTIR is a request bit for the Watchdog Timer input frequency selection, while bit WDTDR is a request bit for the Disable Mode of the WDT. These two bits are only request bits in that they do not actually control the input frequency and disabling of the WDT. They can be modified only when the ENDINIT bit is 0, but they will have an effect only when ENDINIT is properly set to 1 again.

The status register WDT_SR holds information about the current conditions of the WDT. It contains the current timer count value (WDTTIM), three bits indicating the mode of operation (WDTTO for Time-Out Mode, WDTPR for Prewarning Mode, and WDTDS for Disable Mode), and the error indication bits for timer overflow (WDTOE) and access error (WDTAE).

While WDT_SR is a read-only register, the control registers can be read and written. Reading these registers is always possible; a write access, however, must follow certain protocols. Register WDT_CON1 is Supervisor Mode and EndInit-protected, thus, Supervisor Mode must be active and bit ENDINIT must be 0 for a successful write to this register. If one or both conditions are not met, a bus error will be generated, and the bits in WDT_CON1 will be not modified.

Register WDT_CON0 requires a much more complex write procedure as it has a special write protection mechanism. Proper access to WDT_CON0 always requires two write accesses in order to modify its contents. The first write access requires a password to be written to the register to unlock it. This access is called *Password Access*. Then, the second access can modify the register's contents. It is called *Modify Access*. When the modify access completes, WDT_CON0 is locked again automatically. (Even if no parameters are changed in the second write access, it is still called a *modify access*.) If the *Modify Access* sets WDT_CON0.ENDINIT = 0, then other protected system registers, such as WDT_CON1, are unlocked and can be modified.

Note: WDT_CON0 is automatically re-locked after a modify access, so a new password access must be performed to modify it again. Note further that the WDT switches to Time-Out Mode as a side-effect of a successful password access, so that protected registers can remain unlocked at most for the duration of one Time-out Period. Otherwise, the system will be forced to reset.

18.4.2 Modes of the Watchdog Timer

The Watchdog Timer can operate in one of four different modes:

- Time-Out Mode
- Normal Mode
- Disable Mode
- Prewarning Mode

The following description provides a short overview of these modes and how the WDT changes from one mode to the other. As well as these major operating modes, the WDT has special behavior during power-saving and OCDS suspend modes. Detailed discussions of each of the modes can be found in **Section 18.4.6**.

Figure 18-3 provides a state diagram of the different modes of the WDT and the transition possibilities. Please refer to the description for the conditions for changing from one state to the other.

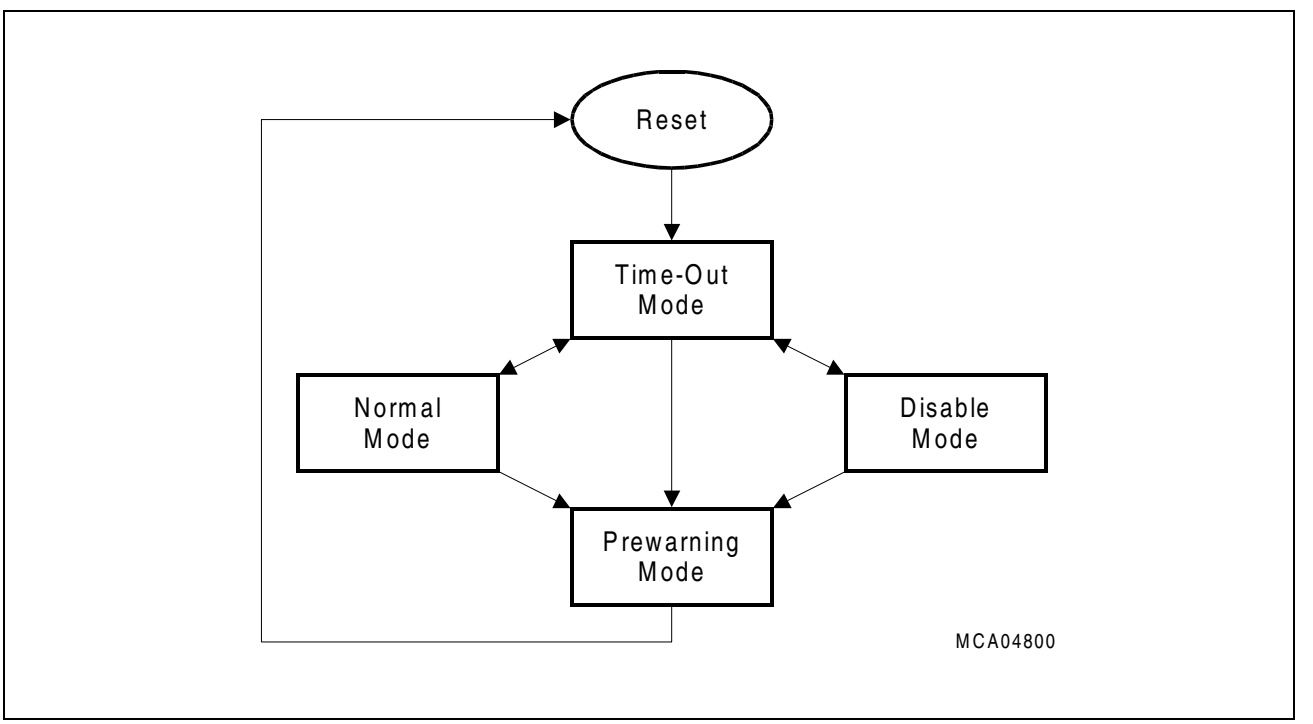


Figure 18-3 State Diagram of the Modes of the WDT

18.4.2.1 Time-Out Mode

The Time-Out Mode is the default mode after a reset. It is also always entered when a valid password access to register WDT_CON0 is performed (see **Section 18.4.3**). The timer is set to a predefined value and starts counting upwards. Time-Out Mode can only be exited properly by setting ENDINIT to one with a correct access sequence. If an improper access to the WDT is performed, or if the timer overflows before ENDINIT is set to 1, a Watchdog Timer NMI request (WDT_NMI) is requested, and Prewarning Mode is entered. A reset of the TC1765 is imminent and can no longer be stopped.

A proper exit from Time-Out Mode can either be to the Normal or the Disable Mode, depending on the state of the disable request bit, WDTDR, in register WDT_CON1.

18.4.2.2 Normal Mode

In Normal Mode (WDTDR = 0), the WDT operates in a standard Watchdog fashion. The timer is set to a user-defined start value, and begins counting up. It has to be serviced before the counter overflows. Servicing is performed through a proper access sequence to the WDT control register WDT_CON0. This reloads the timer with the start value, and normal operation continues.

If the WDT is not serviced before the timer overflows, or if an invalid access to the WDT is performed, a system malfunction is assumed. Normal Mode is terminated, a Watchdog Timer NMI request (WDT_NMI) is requested, and Prewarning Mode is entered. A reset of the TC1765 is imminent and can no longer be stopped.

Because servicing the WDT is an access sequence, first requiring a valid password access to register WDT_CON0, the WDT will enter Time-Out Mode until the second proper access is performed.

18.4.2.3 Disable Mode

Disable Mode is provided for applications which truly do not require the Watchdog Timer function. It can be entered from Time-Out Mode when the disable request bit WDTDR is set to 1. The timer is stopped in this mode. However, disabling the WDT does only stop it from performing the standard Watchdog function (Normal Mode), eliminating the need for timely service of the WDT. It does not disable Time-Out and Prewarning Mode. If an access to register WDT_CON0 is performed in Disable Mode, Time-Out Mode is entered if the access was valid, and Prewarning Mode is entered if the access was invalid. Thus, the ENDINIT monitor function as well as (a part of) the system malfunction detection will still be active.

18.4.2.4 Prewarning Mode

Prewarning Mode is entered always when a Watchdog error is detected. This can be an overflow of the timer in Normal or Time-Out Mode, or an invalid access to register WDT_CON0. Instead of immediately generating a reset of the device, as known from other Watchdog timers, the TC1765 Watchdog Timer provides the system with a chance to save important state information before the reset occurs. This is done through first activating an NMI trap request to the CPU, warning it about the coming reset (reset prewarning). If the CPU is still able to do so (depending on the type and severity of the detected malfunction), it can react on the Watchdog NMI request and can save important system state to memory. This saved system state can then be examined during debugging to determine the cause of the malfunction. If the part would be immediately reset on the detection of a Watchdog error, this debugging information would never be available, and investigating the cause of the malfunction would be a very difficult task.

In Prewarning mode, after having generated the NMI request, the WDT counts for a specified period of time, and then generates a Watchdog reset for the device. This reset generation cannot be avoided in this mode; the WDT does not react anymore to accesses to its registers, nor will it change its state. This is to prevent a malfunction from falsely terminating this mode, disabling the reset, and letting the device to continue to function improperly.

Note: In Prewarning Mode, it is not required for the part waits for the end of this mode and the reset. After having saved required state in the NMI routine, software can execute a soft reset to shorten the time. However, the state of the Watchdog Status Register should also be saved in this case, because the error flags contained in it will be cleared due to the soft reset (this is not the case if the Watchdog reset is awaited).

18.4.3 Password Access to WDT_CON0

A correct password must be written to register WDT_CON0 in order to unlock it for modifications. Software must either know the correct password in advance or it can compute it at runtime. The password required to unlock the register is formed by a combination of bits in registers WDT_CON0 and WDT_CON1, plus a number of guard bits. **Table 18-2** summarizes the requirements for the password.

Table 18-2 Password Access Bit Pattern Requirements

Bit Position	Required Value
0	Current state of the ENDINIT bit, WDT_CON0.ENDINIT
1	Fixed; must be written with 0.
2	Current state of the input frequency request bit, WDT_CON1.WDTIR
3	Current state of the input frequency request bit, WDT_CON1.WDTDR
[7:4]	Fixed; must be written to 1111_B.
[15:8]	Current value of user-definable password field, WDT_CON0.WDTPW
[31:16]	Current value of user-definable reload value, WDT_CON0.WDTREL

When reading register WDT_CON0, bit positions [7:4] always return 0s. As can be seen from **Table 18-2**, the password is designed such that it is not possible to just read the contents of a register and use this as the password. The password is never identical to the contents of WDT_CON0 or WDT_CON1, it is always required to modify the read value (at least bits 1 and [7:4]) to get the correct password. This prevents a malfunction from accidentally reading a WDT register's contents and writing it to WDT_CON0 as an unlocking password.

If the password matches the requirements, WDT_CON0 will be unlocked as soon as the password access has finished. The unlocked condition will be indicated by WDT_CON0.WDTLCK = 0.

If WDT_CON0 is successfully unlocked, a subsequent write access can modify it, as described in **Section 18.4.4**.

If an improper password value is written to WDT_CON0 during the password access, a Watchdog Access Error condition exists. Bit WDTAE is set and the Prewarning Mode is entered.

The user-definable password, WDTPW, provides additional options for adjusting the password requirements to the application's needs. It can be used, for instance, to detect unexpected software loops or to monitor the execution sequence of routines. See **Section 18.5.4**.

18.4.4 Modify Access to WDT_CON0

If WDT_CON0 is successfully unlocked as described in **Section 18.4.3**, the following write access to WDT_CON0 can modify it. However, also this access must follow certain requirements in order to be accepted and regarded as valid. **Table 18-3** lists the required bit patterns. If the access does not follow these rules, a Watchdog Access Error condition is detected, bit WDTAE is set and the Prewarning Mode is entered.

Table 18-3 Modify Access Bit Pattern Requirements

Bit Position	Value
0	User definable; desired value for the ENDINIT bit, WDT_CON0.ENDINIT.
1	Fixed; must be written with 1.
2	Fixed; must be written with 0.
3	Fixed; must be written with 0.
[7:4]	Fixed; must be written with $1111_{B.}$
[15:8]	User-definable; desired value of user-definable password field, WDT_CON0.WDTPW.
[31:16]	User-definable; desired value of user-definable reload value, WDT_CON0.WDTREL.

After the modify access has completed, WDT_CON0.WDTLCK is set to 1 again by hardware, automatically re-locking WDT_CON0. Before the register can be modified again, a valid password access must be executed again.

18.4.5 Term Definitions for WDT_CON0 Accesses

To simplify the descriptions in the following sections, a number of terms are defined to indicate the type of access to register WDT_CON0:

Watchdog Access Sequence: Two accesses to register WDT_CON0 consisting of first a *Password Access* followed by a *Modify Access*. The two accesses do not have to be adjacent accesses, any number of accesses to other addresses can be between these accesses unless the *Time-out Period* is not exceeded.

Password Access: The first access of a *Watchdog Access Sequence* to register WDT_CON0 intended to open WDT_CON0 for modifications. This access needs to write a defined password value to WDT_CON0 in order to successfully open WDT_CON0.

Valid Password Access: A *Password Access* with the correct password value. A Valid Password Access opens register WDT_CON0 for one, and only one, *Modify Access*. Bit WDTLCK is set to 0 after this access. The Watchdog Timer is placed into the Time-Out Mode after a Valid Password Access in Normal Mode or Disabled Mode.

Modify Access: The second access of an *Watchdog Access Sequence* to register WDT_CON0 intended to modify parameters in WDT_CON0. The parameters that can be modified are WDTREL, WDTPW and ENDINIT. Special guard bits in WDT_CON0 must be written with predefined values in order for this access to be accepted.

Valid Modify Access: A *Modify Access* with the correct guard bit values. The values written to WDTREL, WDTPW, and ENDINIT are in effect after completion of this access. Bit WDTLCK is automatically set to 1 after this access. Register WDT_CON0 is locked until it is re-opened with a *Valid Password Access* again.

18.4.6 Detailed Descriptions of the WDT Modes

The following subsections provide detailed descriptions of each of the modes of the WDT. The entry conditions and actions, operation in this mode, as well as exit conditions and the succeeding mode are listed for each mode.

18.4.6.1 Time-Out Mode Details

Time-Out Mode is the default after reset, and is entered each time a Valid Password Access to register WDT_CON0 is performed.

Table 18-4 WDT Time-Out Mode

State / Action	Description
Entry	– Automatically after any reset. – If a valid password was written to WDT_CON0 in Normal or Disable Mode.
Actions on Entry	– WDTTIM is set to $FFFC_H$; WDTTO is set to 1; WDTDS is set to 0. – ENDINIT = 0 if mode entered through reset; otherwise, it retains its previous value. – Bits WDTAE and WDTOE depend on their state before the reset if the reset was caused by the Watchdog. For any other reset (POR, HRST, SRST, PWDRST), they are 0. – WDTIS retains its previous value. – After reset, EndInit is 0. Thus, access to EndInit-protected registers is enabled. If Time-Out Mode was entered through other reasons, ENDINIT might or might not be 0.
Operation	– Timer starts counting up from $FFFC_H$; increments with clock rate determined through WDTIS (0 after reset, slowest clock). – Access to registers WDT_CON0 is possible. Access to register WDT_CON1 is possible if ENDINIT = 0. – Restarting Time-Out Mode is not possible: A valid password access in this mode does not invoke another Time-out sequence (it does not reload the timer, etc.). A modify access to WDT_CON0 writing a 0 to ENDINIT does not terminate Time-Out Mode. – It is not possible to change the reload value or frequency in Time-Out Mode, as this would require setting EndInit to 1, which terminates Time-Out Mode. Reload value is not used until Normal mode is entered.
Exit	a) Writing ENDINIT to 1 with a valid Modify Access (a Valid Password Access must have been executed first). b) Timer WDTTIM overflows from $FFFF_H$ to 0000_H. c) An invalid access to WDT_CON0 (either during the password or the modify access).

Table 18-4 WDT Time-Out Mode

State / Action	Description
Next Mode	Depending on the Exit condition: a1) If WDTDR = 0 (no disable request), the WDT enters the Normal Mode. a2) If WDTDR = 1 (disable request), the WDT enters the Disable Mode. b) Bit WDTOE is set to 1, and the WDT enters the Prewarning Mode. c) Bit WDTAE is set to 1, and the WDT enters the Prewarning Mode.

18.4.6.2 Normal Mode Details

Normal Mode can be entered from Time-Out Mode only if bit WDT_CON1.WDTDR is set to 0 before proper termination of Time-Out Mode. The WDT operates as a standard Watchdog in this mode, requiring timely service to prevent a timer overflow.

Table 18-5 WDT Normal Mode

State / Action	Description
Entry	– Only from Time-Out Mode by writing ENDINIT to 1 with a Valid Modify Access (a Valid Password Access must have been executed first), while bit WDTDR = 0.
Actions on Entry	– WDTTIM is loaded with the value of WDTREL. – Bits WDTAE, WDTOE, WDTPR, WDTTO, and WDTDS are cleared to 0.
Operation	– WDTTIM starts counting up from reload value with frequency selected through WDTIS.
Exit	a) A valid password access to register WDTCON. b) Timer WDTTIM overflows from $FFFF_H$ to 0000_H. c) An invalid access to WDT_CON0 (either during the password or the modify access).
Next Mode	Depending on Exit condition: a) Time-Out Mode. b) Prewarning Mode, bit WDTOE is set to 1 (overflow error). c) Prewarning Mode, bit WDTAE is set to 1 (access error).

18.4.6.3 Disable Mode Details

Disable Mode is provided for applications which truly do not require the Watchdog Timer function. It can only be entered from Time-Out Mode if bit WDT_CON1.WDTDR is set to 1 before proper termination of Time-Out Mode. The counter stops in this mode, eliminating the need for a WDT service. However, if an access to register WDT_CON0 is performed, the WDT will leave Disable Mode. Disable Mode does not stop the detection of access errors and the entry of Prewarning Mode nor the entry of Time-Out Mode on a Valid Password Access.

Table 18-6 WDT Disable Mode

State / Action	Description
Entry	– Only from Time-Out Mode by writing ENDINIT to 1 with a Valid Modify Access (a Valid Password Access must have been executed first), while bit WDTDR = 1.
Actions on Entry	– Bits WDTAE, WDTOE, WDTPR, and WDTTO are cleared. Bit WDTDS is set to 1. – Timer WDTTIM is stopped (it retains its current value).
Operation	–
Exit	a) Valid password access to register WDTCON. b) Invalid access to WDT_CON0 (either during the password or the modify access).
Next Mode	Depending on Exit condition: a) Time-Out Mode. b) Prewarning Mode, bit WDTAE is set to 1 (access error).

18.4.6.4 Prewarning Mode Details

Prewarning Mode is always entered immediately after a Watchdog error condition was detected. This can be either an access error to register WDT_CON0 or an overflow of the counter in Normal or Time-Out Mode. This mode indicates that a reset of the device is imminent. Operation of the WDT in this mode can not be altered or stopped, except through a reset.

Table 18-7 WDT Prewarning Mode

State / Action	Description
Entry	Detection of a Watchdog error: – Overflow of timer WDTTIM. – Access error to register WDT_CON0 (either on a password or modify access) in Time-Out, Normal, or Disable modes.
Actions on Entry	– NMIWDT in register NMISR is set (this triggers an NMI request to the CPU). – WDTTIM is set to $FFFC_H$. – WDTPR is set to 1; WDTDS is set to 0; WDTIS retains its value. – WDTTO retains its previous value: if entry into Prewarning Mode was from Time-Out Mode, WDTTO is 1. In all other cases, WDTTO is 0. – Bits WDTAE and WDTOE indicate whether Prewarning Mode was entered due to an access or an overflow error. They have been set accordingly on exit of the previous mode.
Operation	– Timer WDT_TIM starts counting up from $FFFC_H$ with frequency selected through WDTIS. – Register WDT_CON0 can be accessed in this mode as usual. However, the WDT will not change its mode anymore, regardless whether valid or invalid accesses are made to WDT_CON0. For invalid accesses to WDT_CON0 (password or modify access), however, bit WDTAE in WDT_SR will be set. – Register WDT_CON1 can not be written to in Prewarning Mode, even if bit ENDINIT = 0. Write access to WDT_CON1 is totally prohibited. A write attempt will generate a bus error in this mode.
Exit	– Prewarning Mode can not be disabled, prolonged, or terminated (except through a reset). The timer will increment until it overflows from $FFFF_H$ to 0000_H, which then causes a system reset. Bit WDTRST in register RSTSR is set in this case.
Next Mode	Reset

Note: In Prewarning Mode, it is not required that the part waits for the end of the Time-out Period and the reset. After having saved required state in the NMI routine, software can execute a soft reset to shorten the time. However, the state of the Watchdog Status Register should also be saved in this case, since the error flags contained in it will be cleared due to the soft reset (this is not the case if the Watchdog reset is awaited).

18.4.6.5 WDT Operation During Power-Saving Modes

If the CPU is in Idle Mode, Sleep Mode, or Deep Sleep Mode, it cannot service the Watchdog Timer because no software is running. When in Deep Sleep, only an external event can awaken the system. Excluding this case, and the case where the system is running normally, a strategy for managing the WDT is needed while the CPU is in Idle Mode or Sleep Mode. There are two ways to manage the WDT in these cases. First, the Watchdog can be disabled before idling the CPU. This has the disadvantage that the system will no longer be monitored during the idle period.

A better approach to this problem relies upon a wake-up features of the WDT. Whenever the CPU is put in Idle or Sleep Mode and the WDT is not disabled, it causes the CPU to be awakened at regular intervals. The Watchdog Timer triggers an NMI trap request when its count value (WDT_SR.WDTTIM) transitions from $7FFF_H$ to 8000_H, that is, when the most significant bit of the WDT counter changes its state from 0 to 1. The WDT also sets the NMISR.NMIWDT bit at this time to indicate to the CPU that the WDT caused the NMI. The CPU is awakened by the NMI trap, and can then service the Watchdog Timer in the usual manner, reset NMISR.NMIWDT, and then return to its former power-management mode.

This operation does not cause a WDT error condition. The WDT continues to operate in Normal Mode after generating this wake-up NMI. However, if the CPU does not service the WDT in the NMI trap routine, it will continue to run, eventually causing an overflow, which will cause the WDT to enter Prewarning Mode.

Note: Before switching into a non-running power-management mode, software should perform a Watchdog service sequence. With the Modify Access, the Watchdog reload value, WDT_CON0.WDTREL, should be programmed such that the wake-up occurs after a period which best meets application requirements. The maximum period between NMI requests is one-half of the maximum Watchdog Timer period.

18.4.6.6 WDT Operation in OCDS Suspend Mode

When the On-Chip Debugging System (OCDS) is enabled after reset (through the $\overline{\text{OCDSE}}$ pin), the WDT will automatically stop when OCDS Suspend Mode is activated. It will resume operation after the Suspend Mode is deactivated Watchdog Error.

It is possible that severe system malfunctions may not be corrected even by a system reset. If application code cannot be executed properly because of a system fault, then the WDT initialization code itself might not be able to execute to service the WDT, with the result that two WDT-initiated resets might occur back-to-back. A feature of the WDT detects such Double Watchdog Errors and suspends all system operations after the second reset occurs. This feature prevents the TC1765 from executing random wrong code for longer than the Time-out Period, and prevents the TC1765 from being repeatedly reset by the Watchdog Timer.

The purpose of the Double Watchdog Error feature is to avoid loops such as: Reset - software does not start correctly - prewarning - watchdog reset - software does not start correctly - …

The WDT has an internal counter that generates internally a constant reset after a second watchdog error. This counter can be cleared only by external reset sources (power-on reset or hardware reset).

18.4.7 Determining WDT Periods

The WDT uses the same clock as the System Control Unit (SCU) in which it is integrated. In the TC1765, this clock is equal to the system clock, f_{SYS}. A clock divider in front of the Watchdog Timer provides two output frequencies, $f_{SYS}/256$ and $f_{SYS}/16384$. Bit WDTIS selects between these options.

When the WDT is in Normal Mode, the duration of a WDT cycle is defined as a Normal Period, as described in **Section 18.4.7.2**.

When the WDT is in Time-Out Mode or Prewarning Mode, the duration of a WDT cycle is defined as a Time-out Period, as described in **Section 18.4.7.1**.

The general form to calculate a Watchdog period is:

$$\text{period} = \frac{(2^{16} - \text{startvalue}) \times 256 \times 2^{(1 - \text{WDTIS}) \times 6}}{f_{SYS}}$$

The parameter *startvalue* represents the fixed value $FFFC_H$ for the calculation of the Time-out Period, and the user-programmable reload value WDTREL for the calculation of the Normal Period. Note that the exponent (1 - WDTIS) × 6 results to 0 if WDTIS is 1, and to 6 if WDTIS is 0. This results in the value 256 being multiplied by either 1 ($2^0 = 1$) or by 64 (2^6), giving the two divider factors 256 and 16384.

Note: Because there is no synchronization of the clock divider to the mode transitions of the Watchdog, the next clock pulse, incrementing the counter, may come after one clock divider period, or immediately after the counter was reloaded. Thus, it is recommended that the reload value is programmed to a value which results in one clock pulse more than the required period.

18.4.7.1 Time-out Period

The duration of Time-Out Mode and Prewarning Mode is determined by the Time-out Period described here. The Time-out Period that occurs immediately after reset is governed entirely by system defaults, as no software is been able to run at this point; it is described separately below.

Time-out Period After Reset

After reset, the initial count value for the timer is fixed at $FFFC_H$ when the WDT clock starts running. The WDT counts up at a rate determined by WDT_SR.WDTIS, which is 0 after any reset (f_{SYS}/16384). Counting up from $FFFC_H$, it takes four clocks for the counter to overflow, so the Time-out Period defaults to a period of $4 \times 16384/f_{SYS} = 65536/f_{SYS}$. This establishes the real-time deadline for software to initialize the Watchdog and critical system registers, and to then set ENDINIT. For example, the Time-out Period after reset would correspond to 1.6 ms @ 40 MHz system frequency.

Changing the input frequency selection via WDT_CON1.WDTIR during this initial Time-out Period has no immediate effect, because frequency selection is actually determined by WDT_SR.WDTIS, but WDT_CON1.WDTIR is only copied into WDT_SR.WDTIS after WDT_CON0.ENDINIT has been set to 1, that is, after Time-Out Mode has been properly exited. Hence, the new input frequency will become effective only in a subsequent Time-out Period.

Time-out Period During Normal Operation

As after reset, the WDT counter is initially set to $FFFC_H$ when Time-Out Mode is entered, and Time-Out Mode expires when the counter overflows. However, there are two differences to the Time-out Period after reset. First, the input frequency can be either f_{SYS}/256 or f_{SYS}/16384, depending on the programmed state of bit WDT_SR.WDTIS before the Time-out Period was entered. Second, because there is no synchronization of the clock divider to the mode transitions of the Watchdog, the next clock pulse, incrementing the counter to $FFFD_H$, may come after one clock divider period, or immediately after the counter was initially set to $FFFC_H$. Thus, the minimum duration of the Time-out Period in the latter case will only be three counter clocks. The possible minimum and maximum periods are given in **Table 18-8**.

Table 18-8 Time-out Period During Normal Operation

WDTIS	Min./ Max.	Period	Example @ f_{SYS} = 40 MHz
0	min.	$3 \times 16384/f_{SYS} = 49152/f_{SYS}$	1.2 ms
	max.	$4 \times 16384/f_{SYS} = 65536/f_{SYS}$	1.6 ms
1	min.	$3 \times 256/f_{SYS} = 768/f_{SYS}$	19 µs
	max.	$4 \times 256/f_{SYS} = 1024/f_{SYS}$	26 µs

The WDT input clock rate can not be changed during the Time-out Period. The control bit for the input clock rate, WDT_SR.WDTIS, is loaded from WDT_CON1.WDTIR when WDT_CON0.ENDINIT is set to 1, that is, after Time-Out Mode has been properly exited. Hence, the new input frequency will become effective only in the subsequent Time-out Period.

Note: In Prewarning Mode, it is not required that the part waits for the end of the Time-out Period and the reset. After having saved required state in the NMI routine, software can execute a soft reset to shorten the time. However, the state of the Watchdog Status Register should also be saved in this case, since the error flags contained in it will be cleared due to the soft reset (this is not the case if the Watchdog reset is awaited).

18.4.7.2 Normal Period

The duration of Normal Mode can be varied by two parameters: the input clock and the reload value.

The system clock, f_{SYS}, can be divided by either 256 or 16384. WDT_SR.WDTIS selects the input clock divider. The default value of WDTIS after reset is 0, corresponding to a frequency of f_{SYS}/16384.

When the Watchdog Timer is serviced in Normal Mode, it is reloaded with the 16-bit reload value, WDT_CON0.WDTREL.

The Watchdog Timer Period can be varied over a wide range with these two parameters. Again, since there is no synchronization of the clock divider to the mode transitions of the Watchdog, the next clock pulse, incrementing the counter, may come after one clock divider period, or immediately after the counter was reloaded. Thus, it is recommended that the reload value is programmed to a value which results to one clock pulse more than the required period. Using a reload value of $FFFF_H$ could therefore lead to an immediate overflow of the timer. Thus, the examples given in **Table 18-9** are only shown with a maximum reload value of $FFFE_H$.

Table 18-9 Timer Periods in Normal Mode

WDTIS	Reload Value	Min./ Max.	Period	Example @ f_{SYS} = 40 MHz
0	0000_H	min.	$65535 \times 16384/f_{SYS} = 1073725440/f_{SYS}$	26.8 s
		max.	$65536 \times 16384/f_{SYS} = 1073741824/f_{SYS}$	26.8 s
	$FFFE_H$	min.	$1 \times 16384/f_{SYS} = 16384/f_{SYS}$	410 µs
		max.	$2 \times 16384/f_{SYS} = 32768/f_{SYS}$	819 µs
1	0000_H	min.	$65535 \times 256/f_{SYS} = 16776960/f_{SYS}$	419 ms
		max.	$65536 \times 256/f_{SYS} = 16777216/f_{SYS}$	419 ms
	$FFFE_H$	min.	$1 \times 256/f_{SYS} = 256/f_{SYS}$	6.4 µs
		max.	$2 \times 256/f_{SYS} = 512/f_{SYS}$	12.8 µs

18.4.7.3 WDT Period During Power-Saving Modes

Care needs to be taken when programming the WDT reload value before going to Idle or Sleep Mode. As described in **Section 18.4.6.5**, the state of bit 15 of the Watchdog counter is used to wake up from these modes through a Watchdog NMI request. Thus, the reload value should be chosen such that it is less then $7FFE_H$ (bit 15 = 0), otherwise an immediate wake-up could occur. Only half of the maximum periods shown in **Table 18-9** can be used for the wake-up period.

18.5 Handling the Watchdog Timer

This section describes methods of handling the Watchdog Timer function.

18.5.1 System Initialization

After any reset, the Watchdog Timer is put in Time-Out Mode, and WDT_CON0.ENDINIT is 0, providing access to sensitive system registers. Changes to the operation of the Watchdog Timer controlled by WDT_CON1 become effective only after WDT_CON0.ENDINIT has been set to 1 again. Thus, changes to the WDT mode bits in WDT_CON1 do not interfere with the Time-out operation of the Watchdog Timer after reset. **Table 18-10** shows the default contents of the Watchdog Timer registers.

Table 18-10 Watchdog Timer Default Values After Reset

Register	Default Contents	Description
WDT_CON0	FFFC 0002_H	Reload value is $FFFC_H$, WDTPW is 0; WDT_CON0 is locked (WDTLCK = 1); ENDINIT is 0.
WDT_CON1	$0000\ 0000_H$	Watchdog Timer disable request is 0; input clock request set to f_{SYS}/16384.
WDT_SR	FFFC $001U_H$	The Watchdog counter contains $FFFC_H$ (the initial Time-out value); WDT is operating in Time-Out Mode (WDTTO = 1); WDT is enabled (WDTDS = 0); input clock is f_{SYS}/16384. Bits WDTOE and WDTAE are set to 0 after a power-on, a hard or a soft reset. In case of a reset caused by the WDT, these two bits are set depending on the error condition that caused the Watchdog reset.

Because the Watchdog Timer is in Time-Out Mode after reset, WDT_CON0.ENDINIT must be set to 1 before the Time-out Period expires. This means that initialization of ENDINIT-protected system registers must be complete before the expiration of the Time-out Period, defined in **Section 18.4.7.1**. To set WDT_CON0.ENDINIT to 1, a Valid Password Access to WDT_CON0 must be performed first. During the subsequent Valid Modify Access, WDT_CON0.ENDINIT must be set to 1, which will exit Time-Out Mode. The Watchdog Timer is switched to the operation determined by the new values of WDTIS and WDTDS.

Note: The action described above must absolutely be performed during initialization of the device to properly terminate this mode. Even if the Watchdog function will not be used in an application and the WDT will be disabled, a valid access sequence to the WDT is mandatory. Otherwise, the Watchdog counter will overflow, Prewarning Mode will be entered, and a Watchdog reset will occur at the end of the Time-out Period.

Bit fields WDT_CON0.WDTREL and WDT_CON0.WDTPW can optionally be changed during the Valid Modify Access, but it is not required. WDT_CON0.ENDINIT can be set to 1 or 0, however, setting ENDINIT to 0 does not stop Time-Out Mode. Any values written to WDTREL, WDTPW, and ENDINIT are stored in WDT_CON0, and WDT_CON0 is automatically locked (WDTLCK = 1) after the modify access is finished.

18.5.2 Re-opening Access to Critical System Registers

If some or all of the system's ENDINIT-protected registers must be changed during run time of an application, access can be re-opened. To do this, WDT_CON0 must first be unlocked with a Valid Password Access. In the following Valid Modify Access, ENDINIT can be set to 0. Access to ENDINIT-protected registers is now open again. However, when WDT_CON0 is unlocked, the WDT is automatically switched to Time-Out Mode. Thus, the access window is time-limited. Time-Out Mode is only terminated after ENDINIT has been set to 1 again, requiring another Valid Password and Valid Modify Access to WDT_CON0.

If the WDT is not used in an application and is therefore disabled (WDT_SR.WDTDS = 1), the above described case is the only occasion when WDT_CON0 must be accessed again after the system is initialized. If there are no further changes to critical system registers needed, no further accesses to WDT_CON0, WDT_CON1, or WDT_SR are necessary. However, it is always recommended that the Watchdog Timer be used in an application for safety reasons.

18.5.3 Servicing the Watchdog Timer

If the Watchdog Timer is used in an application and is enabled (WDT_SR.WDTDS = 0), it must be regularly serviced to prevent it from overflowing.

Service is performed in two steps. a Valid Password Access followed by a Valid Modify Access. The Valid Password Access to WDT_CON0 automatically switches the WDT to Time-Out Mode. Thus, the modify access must be performed before the Time-out expires or a system reset will result.

During the following modify access, the strict requirement is that WDT_CON0.ENDINIT as well as bit 1 and bits [7:4] are written with 1's, while bits [3:2] are written with 0's.

Note: ENDINIT must be written with 1 even if it is already set to 1 to perform a proper service.

Changes to the reload value WDTREL, or the user-definable password WDTPW, are not required. However, changing WDTPW is recommended so that software can monitor Watchdog Timer service operations throughout the duration of an application program (see **Section 18.5.4**).

If WDT service is properly executed, Time-Out Mode is terminated, and the Watchdog Timer switches back to its former mode of operation, and Watchdog Timer service is complete.

18.5.4 Handling the User-Definable Password Field

WDT_CON0.WDTPW is an 8-bit field that can be set by software to any arbitrary value during a Modify Access. Settings of this field have no effect on the operation of the WDT, other than the role it plays in forming the password bit pattern, as discussed in **Section 18.4.3**.

The purpose of this field is to support further enhancements to the password protection scheme. For the following description, it is assumed that software does at least not fully compute the value for the Password Access from the contents of registers WDT_CON0 and WDT_CON1, but uses a predefined constant, embedded in the instruction stream, for the password (this is at least necessary for the user-definable password field WDTPW). For example, software can modify this field each time it executes a Watchdog service sequence. The next service sequence needs to take this new value into account for its Password Access. And it again changes the value during its Modify Access. Up to 256 different password values can be used. In this way, each service sequence is unique. If a malfunction occurs that, for instance, would result in the omission of one or more of these service sequences, the next service sequence would most probably not write the correct password. This service sequence would rely on the password value programmed during the normally preceding service sequence. However, if this one was skipped, the password value required by the contents of the Watchdog registers is the one programmed at the last service sequence executed before the malfunction had occurred. A Watchdog error condition would be detected in this case.

In the same manner, the Watchdog would detect the malfunction if a service sequence would be executed twice due to a falsely performed jump. **Figure 18-4** illustrates these examples.

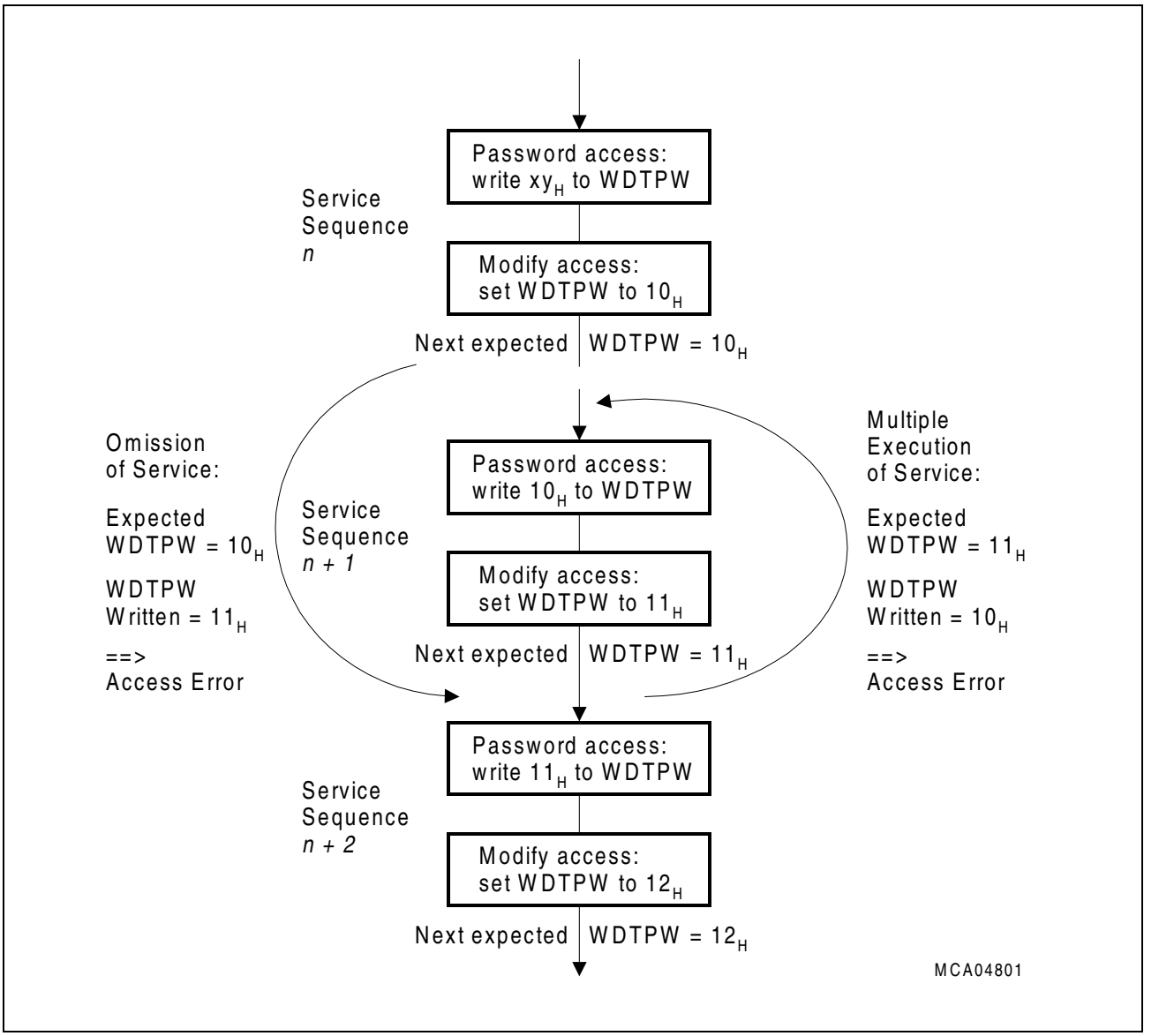


Figure 18-4 Detection of False Jumps and Loops

Other schemes are possible. Consider the case in which a routine determines some conditions that alter the program flow. One of two or more different paths will be executed next depending on these conditions. Before branching to the appropriate routine(s), software performs a Watchdog service and sets the new password value for WDTPW such that it depends on these conditions, that is, some or all of these condition codes can be incorporated into WDTPW. The next service sequence is performed at the point where the different paths come together again. To determine the correct password, software uses a value returned from the path which was executed. This value must match the value in WDTPW, otherwise the wrong path was executed. **Figure 18-5** shows an example for this.

It is also possible to have the different paths of a program compute the full or partial password to unlock register WDT_CON0. The password will only match at the next service sequence if all the expected paths and calculation routines have been executed properly. If one or more steps would have been omitted or a wrong path was executed due to a malfunction, the Watchdog failure mechanism will detect this and issue a reset of the device (after the prewarning phase).

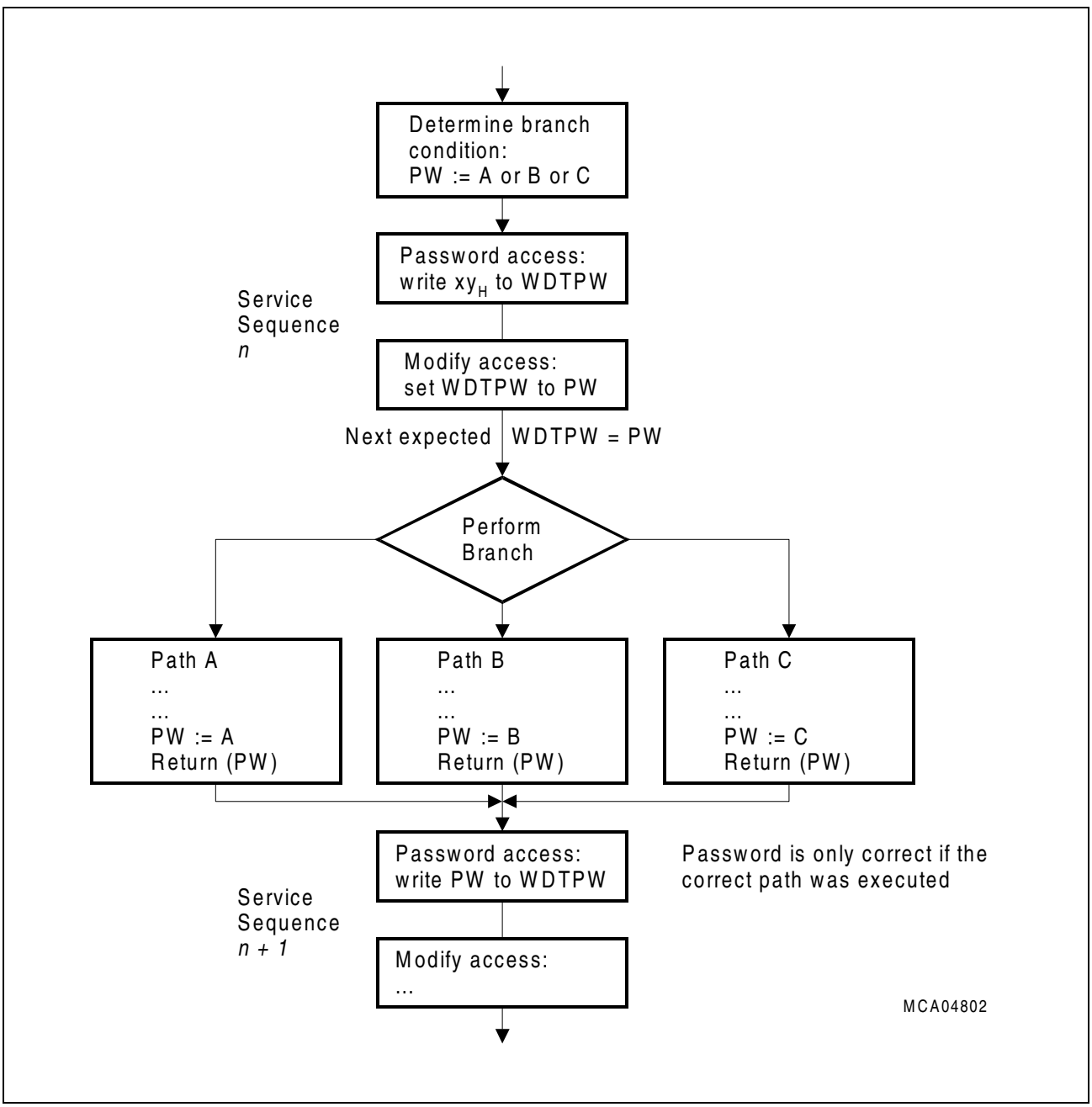


Figure 18-5 Monitoring Program Sequences

18.5.5 Determining the Required Values for a WDT Access

As described in Section 18.4.3 and Section 18.4.4, the values required for the password and modify accesses to register WDT_CON0 are designed such that they can be derived from the values read from registers WDT_CON0 and WDT_CON1. However, at least some bits have to be modified in order to get the correct write value. This makes it very unlikely that a false operation derives values from reading these registers which inadvertently affect the WDT operation when written back to WDT_CON0. Even if a false write operation would have written the correct password to WDT_CON0, one further, different correct value needs to be written to this register in order to have an effect. In addition, the WDT switches to Time-Out Mode after the Valid Password Access, providing only a time-limited window for the second access.

While computing the required values from the current contents of the Watchdog registers is one option, the method of using predetermined values, set at compile-time of the program, may be the better approach in many cases. Usually, handling the Watchdog Timer is performed by one and only one task. Thus, the problem will not occur that another task might have changed some of the parameters which must not be modified (which would require reading the contents, modifying the value appropriately, and then writing it back). The one task handling the Watchdog Timer function would always “know” how it has programmed the WDT last time, and would therefore also “know” the next password value for opening WDT_CON0. In fact, this method would actually detect the case if another task had illegally modified the Watchdog registers, since the predetermined password might not work anymore, and a Watchdog error condition is generated.

In addition, accessing the WDT with predetermined values has the obvious benefit of shorter code, as no computing steps need to be performed.

18.6 Watchdog Timer Registers

Three registers are provided with the Watchdog Timer: WDT_CON0, WDT_CON1, and WDT_SR, as shown in **Figure 18-6**. They are located in the System Control Unit (SCU) Module.

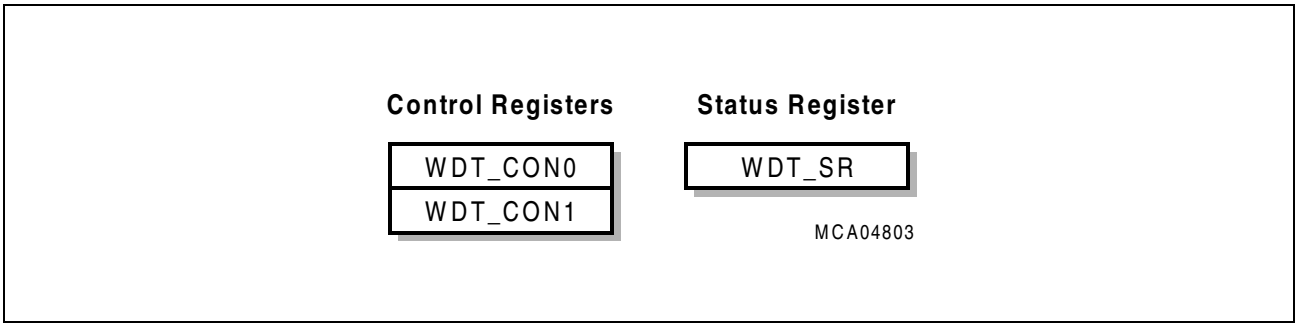


Figure 18-6 Watchdog Registers

Table 18-11 WDT Kernel Registers

Register Short Name	Register Long Name	Offset Address	Description see
WDT_CON0	Watchdog Timer Control Register 0	0020_H	**Page 18-29**
WDT_CON1	Watchdog Timer Control Register 1	0024_H	**Page 18-31**
WDT_SR	Watchdog Timer Status Register	0028_H	**Page 18-32**

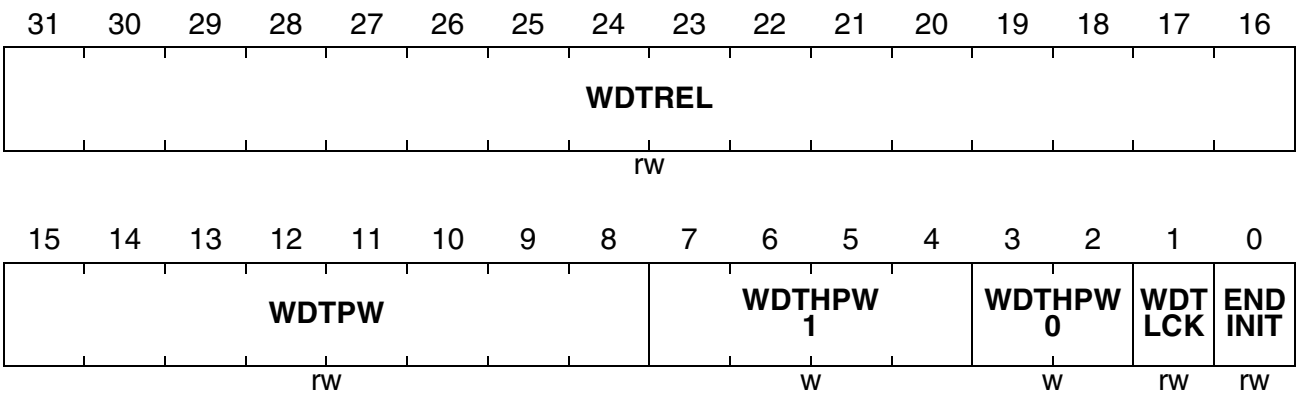
In the TC1765, the registers of the Watchdog Timer are located in the address range of the SCU:

- Module Base Address: F000 0000_H
  Module End Address; F000 $00FF_H$
- Absolute Register Address = Module Base Address + Offset Address

18.6.1 Watchdog Timer Control Register 0

WDT_CON0 manages password access to the Watchdog Timer. It also stores the timer reload value, a user-definable password field, a lock bit, and the end-of-initialization (ENDINIT) control bit.

WDT_CON0
Watchdog Timer Control Register 0 **Reset Value: FFFC 0002$_H$**



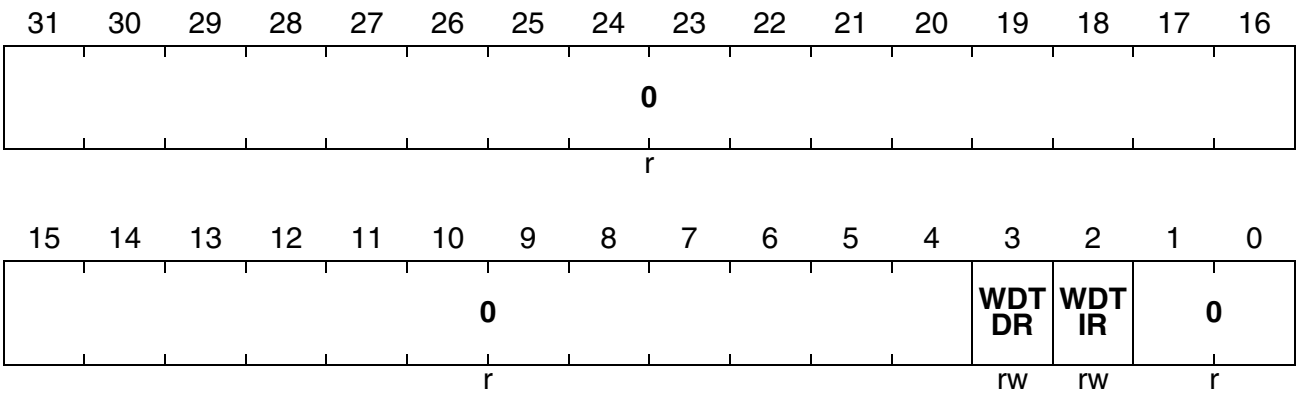
Field	Bits	Type	Description
ENDINIT	0	rw	**End-of-Initialization Control Bit** 0 Access to Endinit-protected registers is permitted (default after reset). 1 Access to Endinit-protected registers is not permitted. ENDINIT controls the access to critical system registers. During a password access it must be written with its current value. It can be changed during a modify access to WDT_CON0.

Field	Bits	Type	Description
WDTLCK	1	rw	**Lock Bit to Control Access to WDT_CON0** 0 Register WDT_CON0 is unlocked. 1 Register WDT_CON0 is locked (default after reset). The actual value of WDTLCK is controlled by hardware. It is set to 0 after a successful password access to WDT_CON0 and automatically set to 1 again after a successful modify access to WDT_CON0. During a write to WDT_CON0 the value written to this bit is only used for the password-protection mechanism and is not stored. This bit must be set to 0 during a password access to WDT_CON0 and set to 1 during a modify access to WDT_CON0. That is, the inverted value read from WDTLCK always must be written to itself.
WDTHPW0	[3:2]	w	**Hardware Password 0** This field must be written with the value of the bits WDT_CON1.WDTDR and WDT_CON1.WDTIR during a password access. This field must be written with 0's during a modify access to WDT_CON0. When read, these bits always return 0.
WDTHPW1	[7:4]	w	**Hardware Password 1** This field must be written to 1111_B during both, a password access and a modify access to WDT_CON0. When read, these bits always return 0.
WDTPW	[15:8]	rw	**User-Definable Password Field for Access to WDT_CON0** This bit field must be written with its current contents during a password access. It can be changed during a modify access to WDT_CON0.
WDTREL	[31:16]	rw	**Reload Value for the Watchdog Timer** If the Watchdog Timer is enabled and in Normal Timer Mode, it will start counting from this value after a correct Watchdog service. This field must be written with its current contents during a password access. It can be changed during a modify access to WDT_CON0 ($FFFC_H$ = default after reset).

18.6.2 Watchdog Timer Control Register 1

WDT_CON1 manages operation of the WDT. It includes the disable request and frequency selection bits. It is ENDINIT-protected.

WDT_CON1
Watchdog Timer Control Register 1 **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
WDTIR	2	rw	**Watchdog Timer Input Frequency Req. Control Bit** 0 Request to set input frequency to f_{SYS}/16384 1 Request to set input frequency to f_{SYS}/256 This bit can only be modified if WDT_CON0.ENDINIT is set to 0. WDT_SR.WDTIS is updated by this bit only when ENDINIT is set to 1 again. As long as ENDINIT is left at 0, WDT_SR.WDTIS controls the current input frequency of the Watchdog Timer. When ENDINIT is set to 1 again, WDT_SR.WDTIS is updated with the state of WDTIR.
WDTDR	3	rw	**Watchdog Timer Disable Request Control Bit** 0 Request to enable the Watchdog Timer. 1 Request to disable the Watchdog Timer. This bit can only be modified if WDT_CON0.ENDINIT is set to 0. WDT_SR.WDTDS is set to this bit's value when ENDINIT is set to 1 again. As long as ENDINIT is left at 0, bit WDT_SR.WDTDS controls the current enable/disable status of the Watchdog Timer. When ENDINIT is set to 1 again with a valid modify access, WDT_SR.WDTDS is updated with the state of WDTDR.

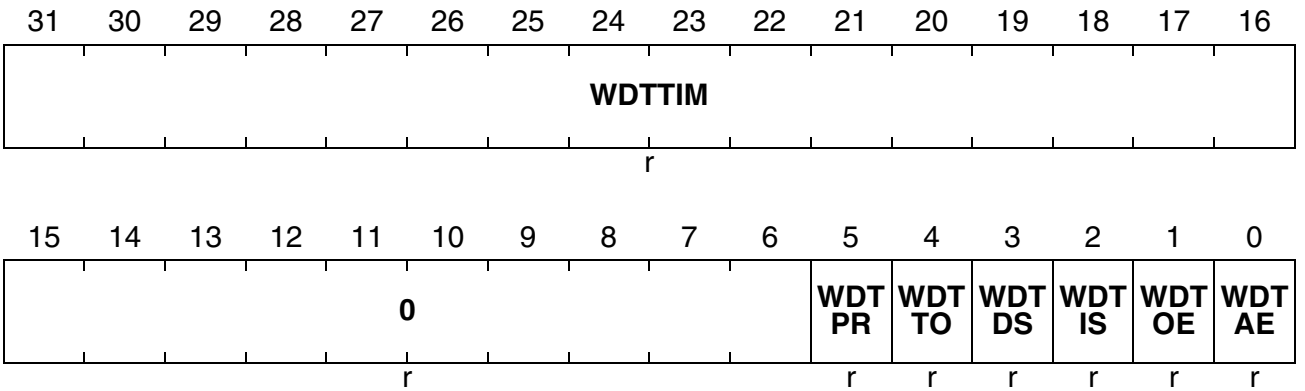
Field	Bits	Type	Description
0	[1:0], [31:4]	r	**Reserved**; read as 0; should be written with 0;

18.6.3 Watchdog Timer Status Register

WDT_SR shows the current state of the WDT. Status include bits indicating reset prewarning, Time-out, enable/disable status, input clock status, and access error status.

The reset value for this register is depending on the cause of the reset. For any reset other than a Watchdog reset, the reset value is FFFC 001U_H. After a Watchdog reset, bits WDTAE and WDTOE indicate the type of Watchdog error which occurred before the Watchdog reset. Either one or both bits can be set. These bits are not reset on a Watchdog reset. Bits WDTDS and WDTIS are always 0 after any reset.

WDT_SR
Watchdog Timer Status Register **Reset Value: FFFC 0010$_H$**



Field	Bits	Type	Description
WDTAE	0	r	**Watchdog Access Error Status Flag** 0 No Watchdog access error. 1 An Watchdog access error has occurred. This bit is set by hardware when an illegal password access or modify access to register WDT_CON0 was attempted. This bit is only reset through: – a power-on, hardware, or software reset occurs – WDT_CON0.ENDINIT is set to 1 during a valid modify access. However it is not possible to reset this bit if the WDT is in Prewarning Mode, indicated by WDT_SR.WDTPR = 1.

Field	Bits	Type	Description
WDTOE	1	r	**Watchdog Overflow Error Status Flag** 0 No Watchdog overflow error. 1 A Watchdog overflow error has occurred. This bit is set by hardware when the Watchdog Timer overflows from $FFFF_H$ to 0000_H. This bit is only reset when: – a power-on, hardware, or software reset occurs; – WDT_CON0.ENDINIT is set to 1 during a valid modify access. However it is not possible to reset this bit if the Watchdog Timer is in Prewarning Mode, indicated by WDT_SR.WDTPR = 1.
WDTIS	2	r	**Watchdog Input Clock Status Flag** 0 Watchdog Timer input clock is f_{SYS}/16384 (default after reset). 1 Watchdog Timer input clock is f_{SYS}/256. This bit is updated with the state of bit WDT_CON1.WDTIR after WDT_CON0.ENDINIT is written with 1 during a valid modify access to register WDT_CON0.
WDTDS	3	r	**Watchdog Enable/Disable Status Flag** 0 Watchdog Timer is enabled (default after reset). 1 Watchdog Timer is disabled. This bit is updated with the state of bit WDT_CON1.WDTDR after WDT_CON0.ENDINIT is written with 1 during a valid modify access to register WDT_CON0.
WDTTO	4	r	**Watchdog Time-Out Mode Flag** 0 Normal mode. 1 The Watchdog is operating in Time-Out Mode (default after reset). This bit is set to 1 when Time-Out Mode is entered, automatically after a reset and after every password access to register WDT_CON0. It is automatically reset by hardware when Time-Out Mode is properly terminated through a valid modify access to WDT_CON0. It is left set when a Watchdog error occurs during Time-Out Mode, and Prewarning Mode is entered.

Field	Bits	Type	Description
WDTPR	5	r	**Watchdog Prewarning Mode Flag** 0 Normal mode (default after reset). 1 The Watchdog is operating in Prewarning Mode. This bit is set to 1 when a Watchdog error is detected. The Watchdog Timer has issued an NMI trap and is in Prewarning Mode. A reset of the chip occurs after the prewarning period has expired.
WDTTIM	[31:16]	r	**Watchdog Timer Value** Reflects the current content of the Watchdog Timer.
0	[15:6]	r	**Reserved**; read as 0;

19 On-Chip Debug Support

The On-Chip Debug Support (OCDS) of the TC1765 consists of four building blocks:

- OCDS Module in the TriCore CPU (see **Section 19.1**)
- Trace Module (see **Section 19.2**)
- DMA Controller Trace (see **Section 19.3**)
- Debugger Interface Cerberus (see **Section 19.4**)

Figure 19-1 shows a basic block diagram of the building blocks.

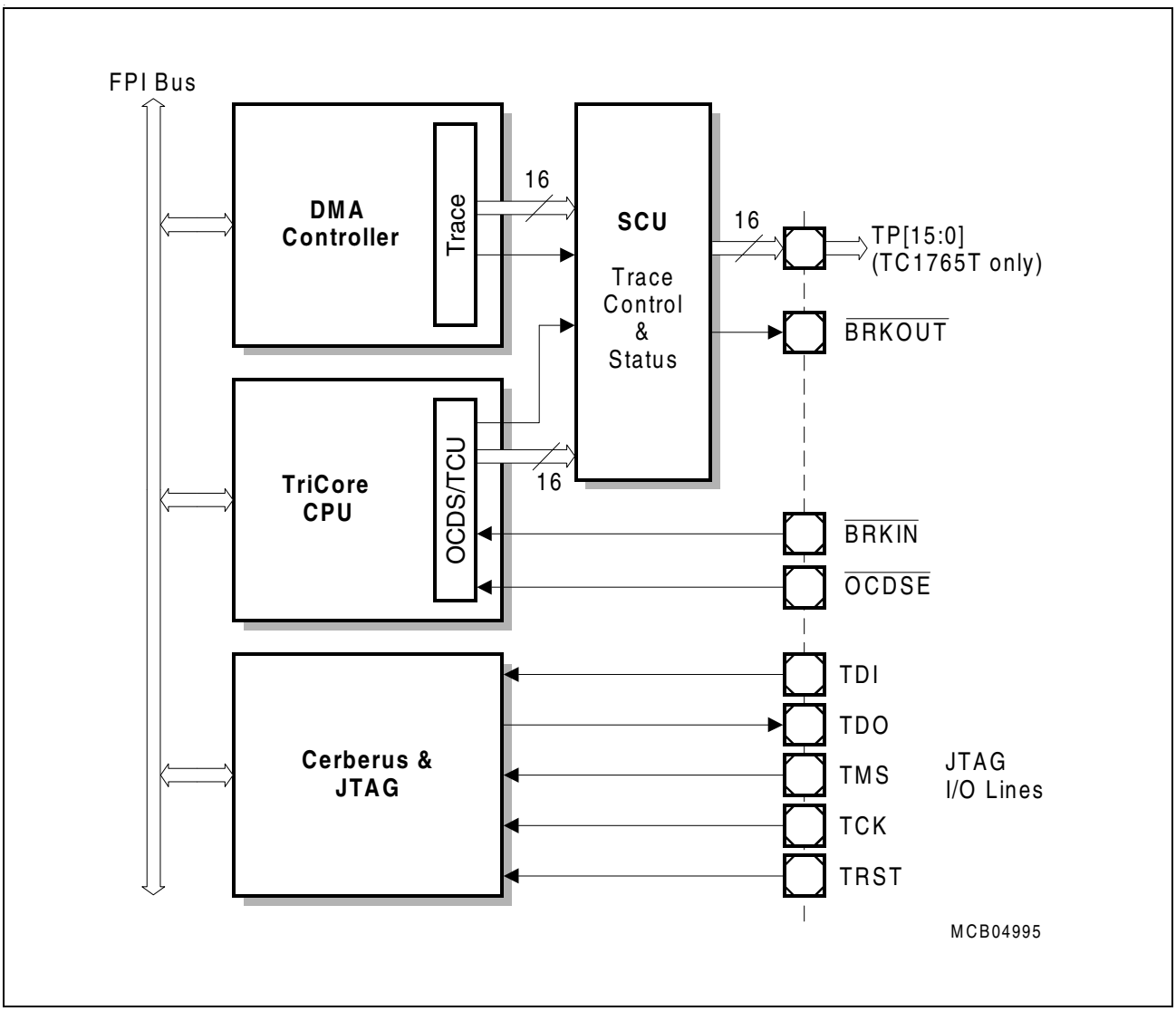


Figure 19-1 OCDS Basic Block Diagram

19.1 TriCore CPU Debug Support

The TriCore CPU in the TC1765 provides On-Chip Debug Support (OCDS) with the following features:

- On-chip breakpoint hardware
- Support of an external break signal

19.1.1 Basic Concepts

The TriCore breakpoint concept has two parts. The first part defines the generation of debug events and the second part defines what actions are taken when a debug event is generated.

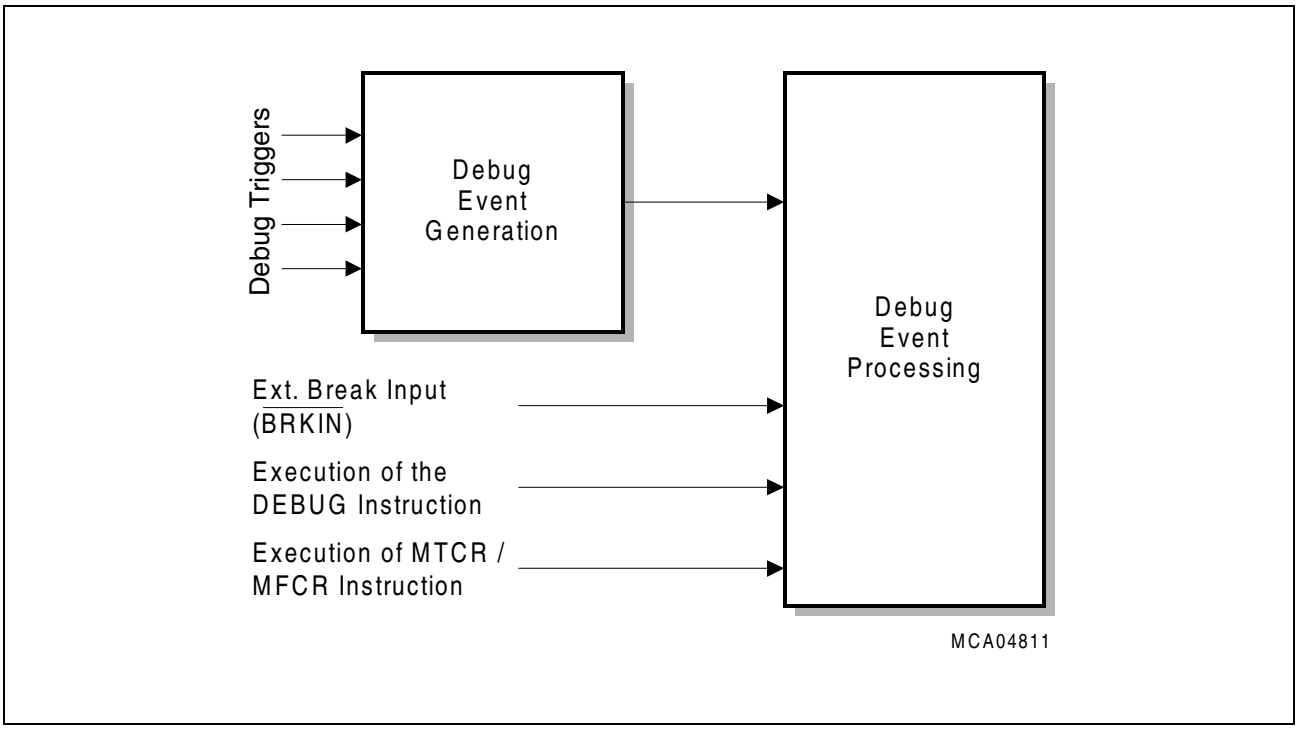


Figure 19-2 Basic TriCore Debug Concept

19.1.2 Debug Event Generation

In order for any debug event to be generated, the debug enable bit DBGSR.DE in the Debug Status Register must be set. If this bit is set, debug events can be generated by the:

– An active (low) signal at the OCDS Break Input pin $\overline{\text{BRKIN}}$
– Execution of a debug instruction
– Execution of a MTCR/MFCR instruction
– Debug event generation unit

19.1.2.1 External Debug Break Input

An external debug break pin is provided to allow the emulator to interrupt the processor asynchronously. The action that is performed when the external debug break input is activated is defined by the contents of the External Break Input Event Specifier Register EXEVT.

Note: The CPU core detects the active edge of $\overline{BRKIN}$ and performs the action specified in EXEVT at the first available opportunity.

19.1.2.2 Software Debug Event Generation

The TriCore architecture also supports a mechanism through which software can explicitly generate a debug event. This can be used, for instance, by a debugger to patch code held in RAM in order to implement breakpoints. A special DEBUG instruction is defined which is a user mode instruction, and its operation depends on whether the debug mode is enabled.

If debug mode is enabled (DBGSR.DE = 1), the DEBUG instruction causes a debug event to be raised and the action defined in the Software Break Event Specifier Register SWEVT is taken. If the debug mode is not enabled, then the DEBUG instruction is treated as a NOP instruction.

Both 16-bit and 32-bit forms of the DEBUG instruction are provided.

19.1.2.3 Execution of a MTCR or MFCR Instruction

In order to protect the emulator resource, a debug event is raised whenever a MTCR or MFCR instruction is used to read or modify an user core SFR. That means that an event is not raised when the user reads or modifies one of the dedicated debug core SFRs.

- DBGSR or
- CREVT or
- SWEVT or
- EXEVT or
- TR0EVT or
- TR1EVT

The action that is performed when a MTCR or MFCR instruction is executed on user core SFRs defined by the content of the Emulator Resource Protection Event Specifier Register CREVT.

19.1.2.4 Debug Event Generation from Debug Triggers

- The debug event generation unit is responsible for generating debug events when a programmable set of debug triggers are active.
- Code protection logic
- Data protection logic

These debug triggers provide the inputs to a programmable block of combinational logic that outputs debug events.

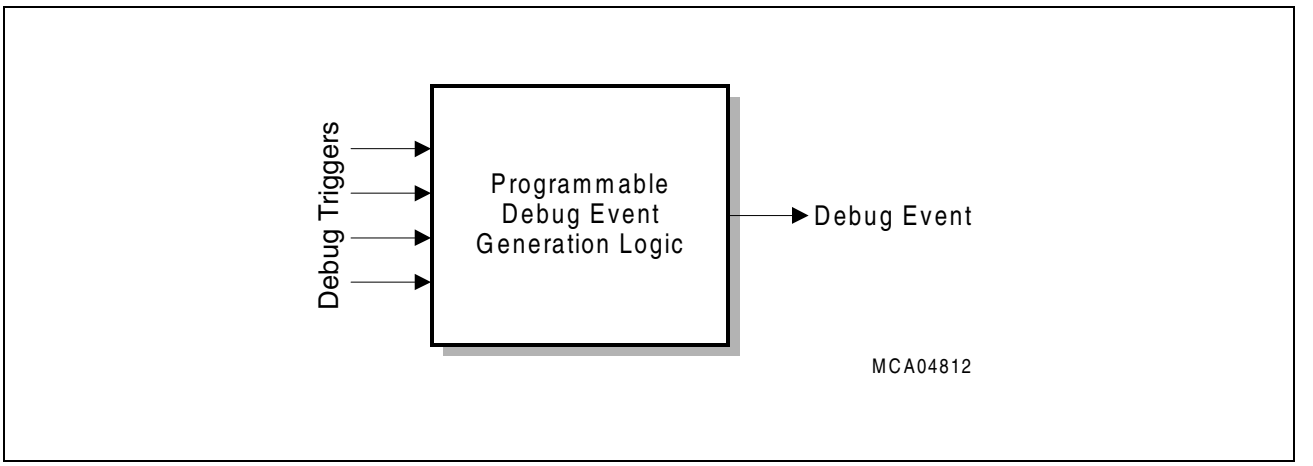


Figure 19-3 Debug Event Generation Logic

The aim is to be able to specify the breakpoints which use fairly simple criteria purely in the on-chip debug event generation unit, and to rely on help from the external debug system or debug monitor to implement more complex breakpoints.

19.1.3 Debug Triggers

19.1.3.1 Protection Mechanism

The TriCore debug system is also integrated into the protection mechanism which can generate the following types of debug triggers:

- Trigger on execution of an instruction at a specific address
- Trigger on execution of an instruction within a range of addresses
- Trigger on the loading of a value from a specific address
- Trigger on the loading of a value from anywhere in a range of addresses
- Trigger on the storing of a value to a specific address
- Trigger on the storing of a value to anywhere in a range of addresses

Informations on the generation of debug triggers by the protection mechanism are given in the TriCore Architecture Manual.

19.1.3.2 Combination of Triggers

In the TC1765 the first two code and data ranges can be used to generate one debug event. The TriCore CPU in the TC1765 allows one code range and one data range to be combined for a debug event generation. The combination is controlled by the Trigger Event n Specifier Registers TRnEVT (n = 0, 1).

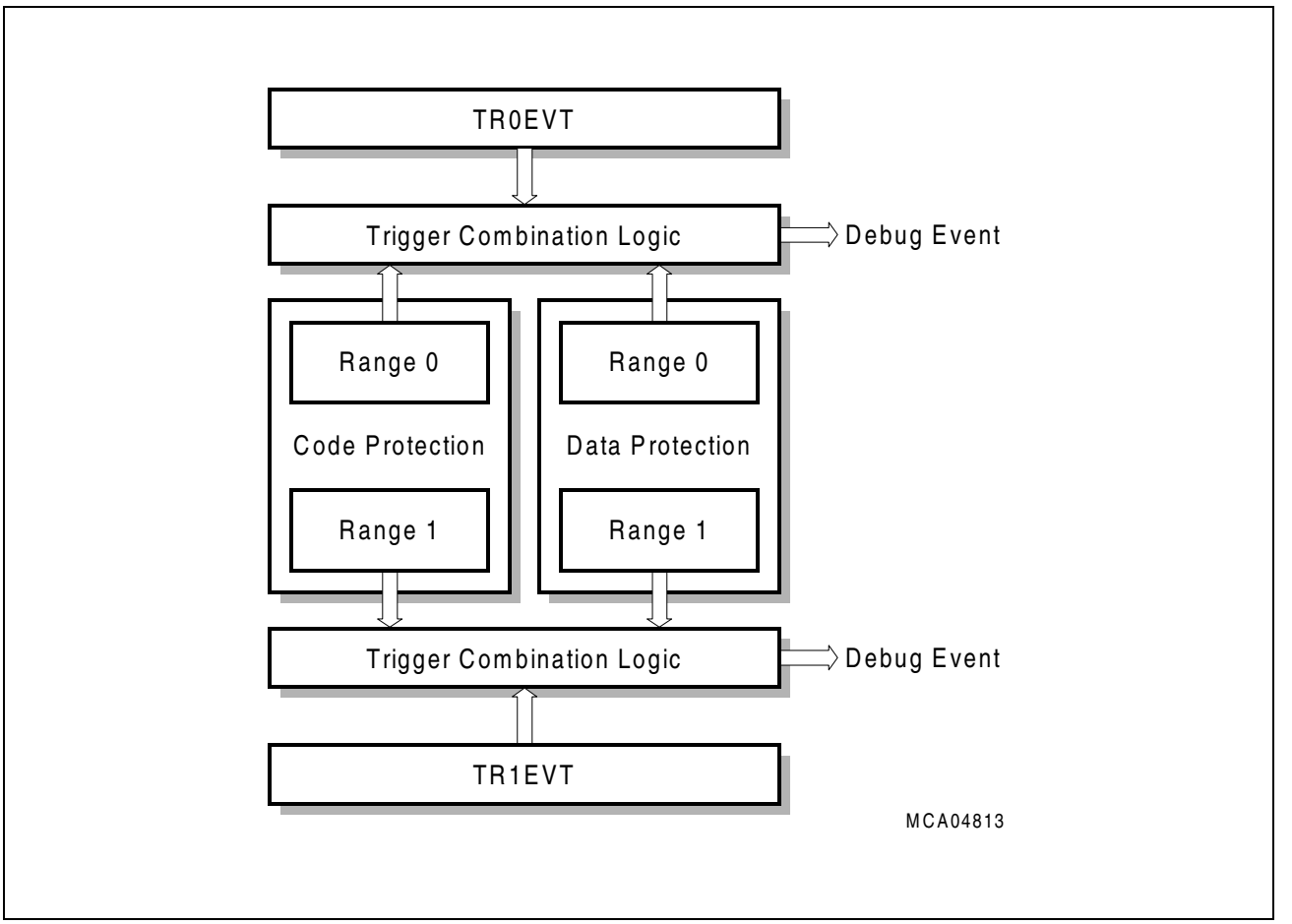


Figure 19-4 Combination of Data and Code Triggers

For example, the inputs from range 0 of the code protection logic can be combined with the inputs from range 0 of the data protection logic. This combination and the action taken if a debug event is generated are controlled by the TR0EVT register.

The debug event generation logic places certain restrictions on which debug triggers can be combined in order to produce a debug event whose action is marked as “break before make”.

All debug events that are produced from a combination of triggers which include inputs from the data protection logic are treated as “break after make”, irrespective of the event specifier.

19.1.4 Actions Taken on a Debug Event

When a debug event is generated, one of the following actions is taken.

19.1.4.1 Assert an External Pin $\overline{\text{BRKOUT}}$

A signal can be asserted on the external pin $\overline{\text{BRKOUT}}$. This could be used in critical routines where the system cannot be interrupted to signal to the external world that a particular event has happened. This feature could also be useful to synchronize the internal and external debug hardware.

For example, when the CPU writes to an off-chip location through the external bus interface, this could be detected and the external pin asserted. This could then be used as the input trigger to an analyzer to capture the bus cycles on the external interface pins.

19.1.4.2 Halt

The halt mode performs a selective cancellation of:

- All instruction after and including the instruction that caused the breakpoint if EXEVT.BBM = 1
- All instructions after the instruction that caused the breakpoint if EXEVT.BBM = 0

Once the pipeline has been cancelled, it enters a halt mode where no more instructions are executed. It then relies on the external debug system to interrogate the target purely through the mapping of the architectural state into the FPI address space without any help from the core.

While halted, the core will not respond to any interrupts, and will only resume execution once the external debug hardware clears the halt bit by writing 10_B to the DBGSR.HALT bit field.

When the halt mode is entered, the following actions are also performed:

- The DBGSR.EVTSRC bit field is updated.
- The breakout pin $\overline{\text{BRKOUT}}$ is asserted for one cycle.

19.1.4.3 Breakpoint Trap

The breakpoint trap is designed to be used to enter a debug monitor without using any user resource. It relies upon the following emulator resources:

- The debug monitor is held in the emulator region at address BE00 0000_H
- There is a 4-word area of RAM available at address BE80 0000_H which can be used to store critical state during the debug monitor entry sequence.

When a breakpoint trap is taken, the following actions are performed:

- Write PSW to BE80 0000_H
- Write PCXI to BE80 0004_H
- Write A10 to BE80 0008_H

- Write A11 to BE80 000C$_H$
- A11 = breakpoint PC
- PC = BE00 0000$_H$
- PSW.PRS = 0
- PSW.IO = 2
- PSW.GW = 0
- PSW.IS = 1
- ICR.IE = 0

The corresponding return sequence is provided through the RFM (return from monitor) instruction. This effectively perform the reverse of the above:

- Branch to A11
- Restore PSW from BE80 0000$_H$
- Restore PCXI from BE80 0004$_H$
- Restore A10 from BE80 0008$_H$
- Restore A11 from BE80 000C$_H$

This provides an automated route into the debug monitor which does not use any user resource. The RFM instruction is then used to return control the original task.

When the debug trap is taken, the following actions are also performed:

- The EVTSRC bit field in DBGSR is updated.
- The $\overline{\text{BRKOUT}}$ pin is asserted for one cycle.

19.1.4.4 Software Breakpoint

When a debug event is raised, the system can enter the software debug mode. The software debug mode is basically an interrupt. The software breakpoint interrupt is controlled in the Software Breakpoint Service Request Control Register SBSRC0.

19.1.5 OCDS Registers

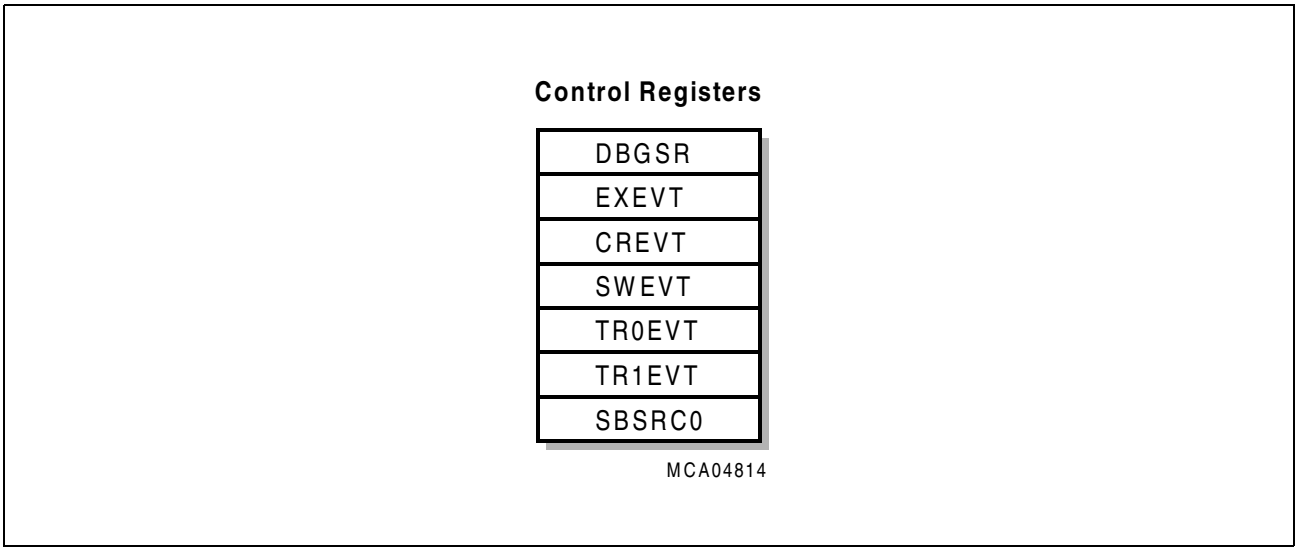


Figure 19-5 OCDS Registers

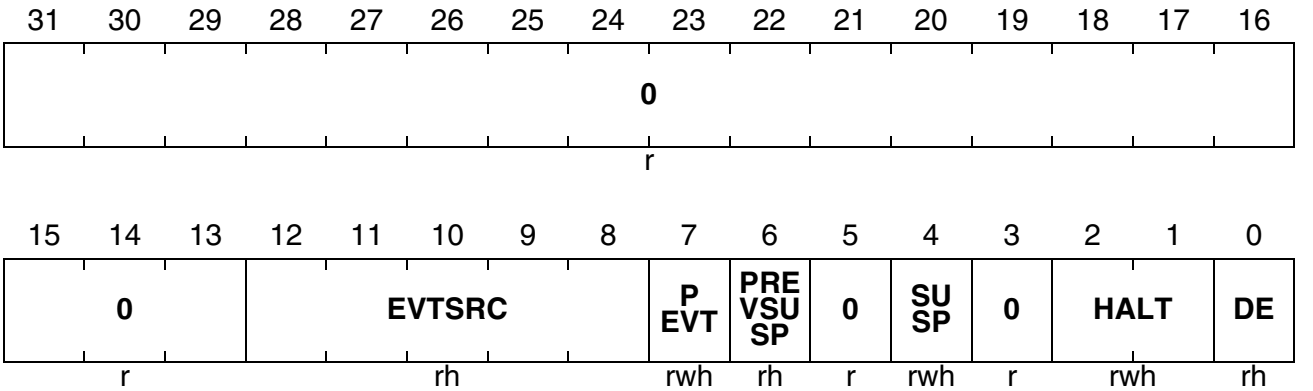
Table 19-1 OCDS Registers

Register Short Name	Register Long Name	Offset Address	Description see
DBGSR	Debug Status Register	0000_H	Page 19-9
EXEVT	External Break Input Event Specifier Register	0008_H	Page 19-11
CREVT	Emulator Resource Protection Event Specifier Register	$000C_H$	Page 19-12
SWEVT	Software Break Event Specifier Register	0010_H	Page 19-13
TR0EVT	Trigger Event 0 Specifier Register	0020_H	Page 19-14
TR1EVT	Trigger Event 1 Specifier Register	0024_H	Page 19-14
SBSRC0	Software Breakpoint Service Request Control Register 0	$00BC_H$ [1)]	Page 19-15

[1)] The SBSRC0 register is located in the address range of the CPU slave interface CPS (see Section 19.6).

DBGSR
Debug Status Register

Reset Value: 0000 0000$_H$

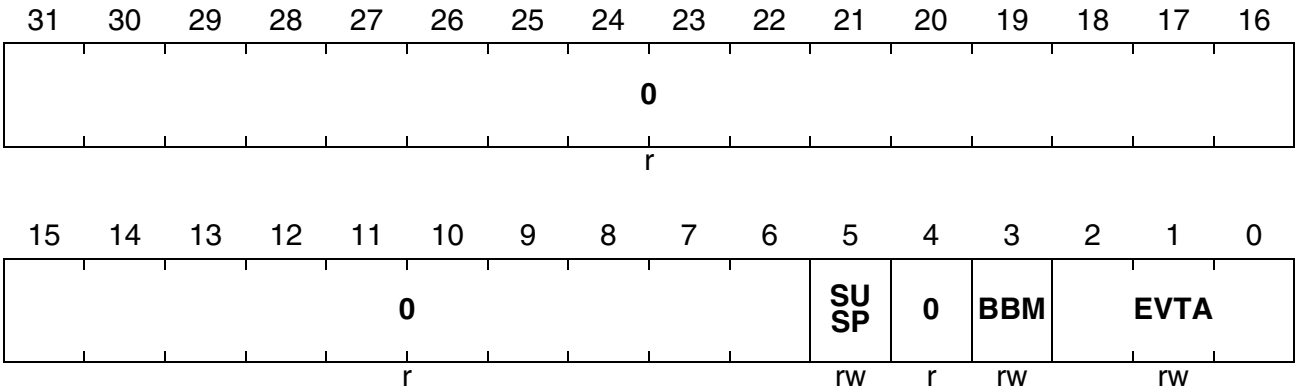


Field	Bits	Type	Description
DE	0	rh	**Debug Enable** Indicates whether debug support was enabled at reset. 0 Debug disabled 1 Debug enabled
HALT	[2:1]	rwh	**CPU Halt Request / Status Field** HALT can be set or cleared by software. HALT[0] is the actual halt bit. HALT[1] is a mask bit to specify whether HALT[0] is to be updated on a software write or not. HALT[1] is always read as 0. HALT[1] must be set to one in order to update HALT[0] by software (R: read; W: write). 00 R: CPU running / W: HALT[0] unchanged 01 R: CPU halted / W: HALT[0] unchanged 10 R: n.a. / W: reset HALT[0] 11 R: n.a. / W: if debug support is enabled (DE = 1), set HALT[0]; if debug support is not enabled (DE = 0), HALT[0] is left unchanged
SUSP	4	rwh	**Current State of the Suspend Signal** 0 Suspend inactive 1 Suspend active
PREVSUSP	6	rh	**Previous State of the Suspend Signal** 0 Previous suspend inactive 1 Previous suspend active
PEVT	7	rwh	**Posted Event** 0 No posted event 1 Posted event

Field	Bits	Type	Description
EVTSRC	[12:8]	rh	**Event Source** 0 EXTEVT 1 CREVT 2 SWEVT 16+n TRnEVT (n = 0, 1) other Reserved
0	3, 5, [31:13]	r	**Reserved**; read as 0; should be written with 0.

EXEVT
External Break Input Event Specifier Register

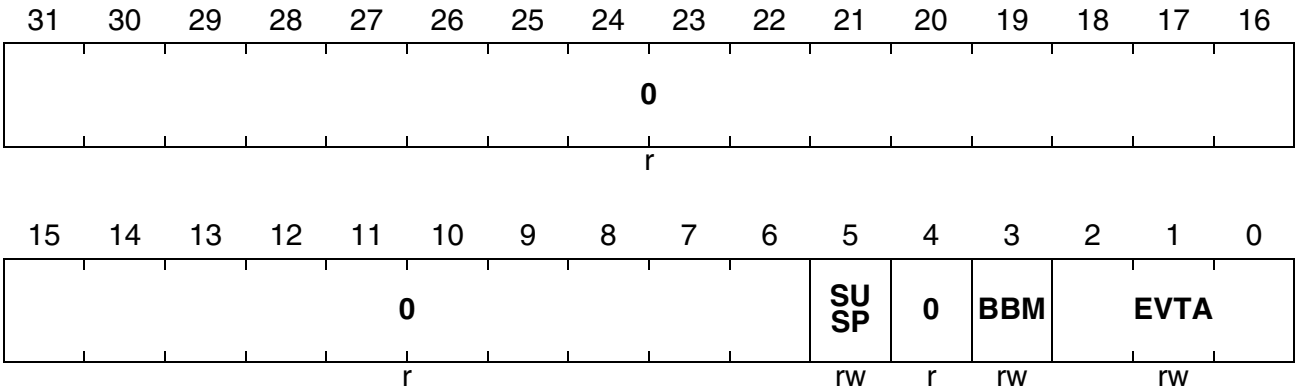
Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
EVTA	[2:0]	rw	**Event Associated** Specifies the action associated with the event: 000 None; disabled 001 Assert external pin $\overline{\text{BRKOUT}}$ 010 Halt 011 Breakpoint trap 100 Software breakpoint 0 101 Reserved, same behavior as 000 110 Reserved, same behavior as 000 111 Reserved, same behavior as 000
BBM	3	rw	**Break Before Make or Break After Make Selection** 0 Break after make 1 Break before make
SUSP	5	rw	**OCDS Suspend Signal State** Value to be assigned to the OCDS suspend signal when the event is raised.
0	4, [31:6]	r	**Reserved**; read as 0; should be written with 0.

CREVT

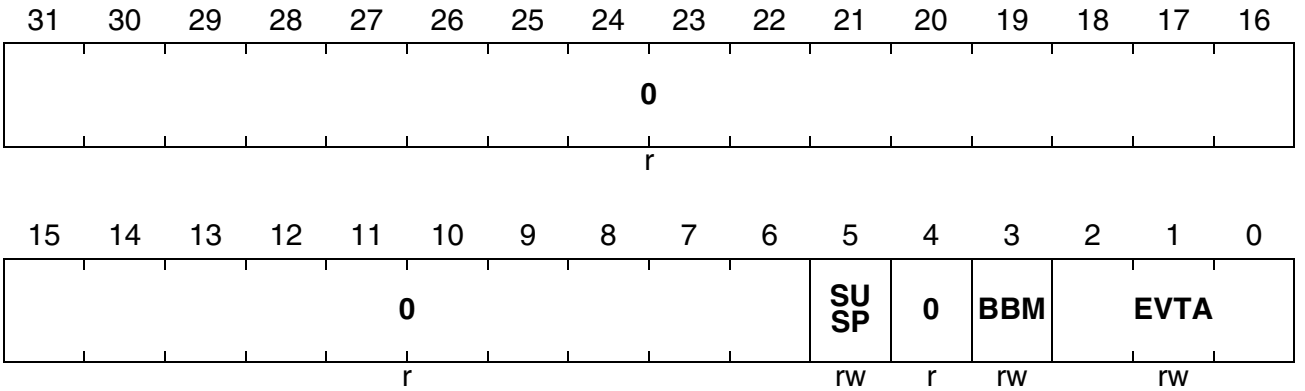
Emulator Resource Protection Event Specifier Register Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
EVTA	[2:0]	rw	**Event Associated** Specifies the action associated with the event: 000 None; disabled 001 Assert external pin $\overline{\text{BRKOUT}}$ 010 Halt 011 Breakpoint trap 100 Software breakpoint 0 101 Reserved, same behavior as 000 110 Reserved, same behavior as 000 111 Reserved, same behavior as 000
BBM	3	rw	**Break Before Make or Break After Make Selection** 0 Break after make 1 Break before make
SUSP	5	rw	**OCDS Suspend Signal State** Value to be assigned to the OCDS suspend signal when the event is raised.
0	4, [31:6]	r	**Reserved**; read as 0; should be written with 0.

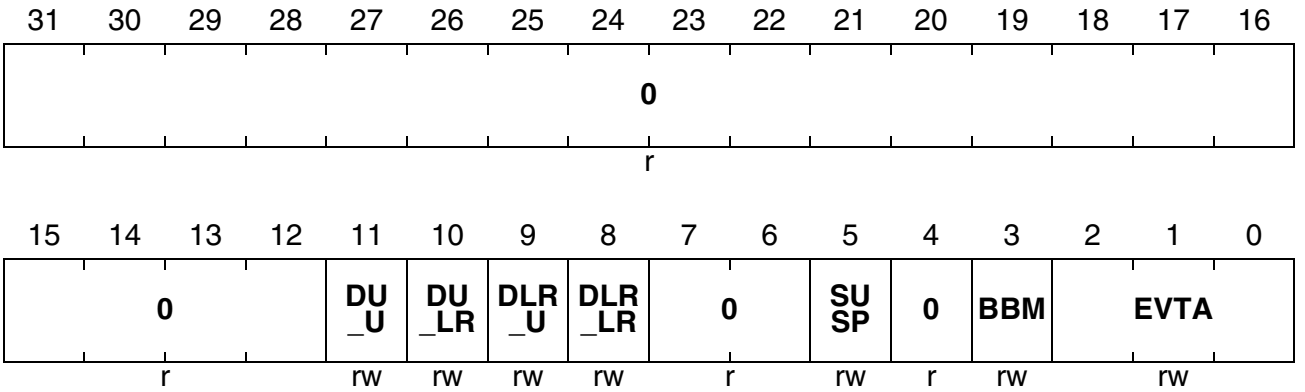
SWEVT
Software Break Event Specifier Register

Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
EVTA	[2:0]	rw	**Event Associated** Specifies the action associated with the event: 000 None; disabled 001 Assert external pin $\overline{\text{BRKOUT}}$ 010 Halt 011 Breakpoint trap 100 Software breakpoint 0 101 Reserved, same behavior as 000 110 Reserved, same behavior as 000 111 Reserved, same behavior as 000
BBM	3	rw	**Break Before Make or Break After Make Selection** 0 Break after make 1 Break before make
SUSP	5	rw	**OCDS Suspend Signal State** Value to be assigned to the OCDS suspend signal when the event is raised.
0	4, [31:6]	r	**Reserved**; read as 0; should be written with 0.

TR0EVT
Trigger Event 0 Specifier Register **Reset Value: 0000 0000$_H$**
TR1EVT
Trigger Event 1 Specifier Register **Reset Value: 0000 0000$_H$**

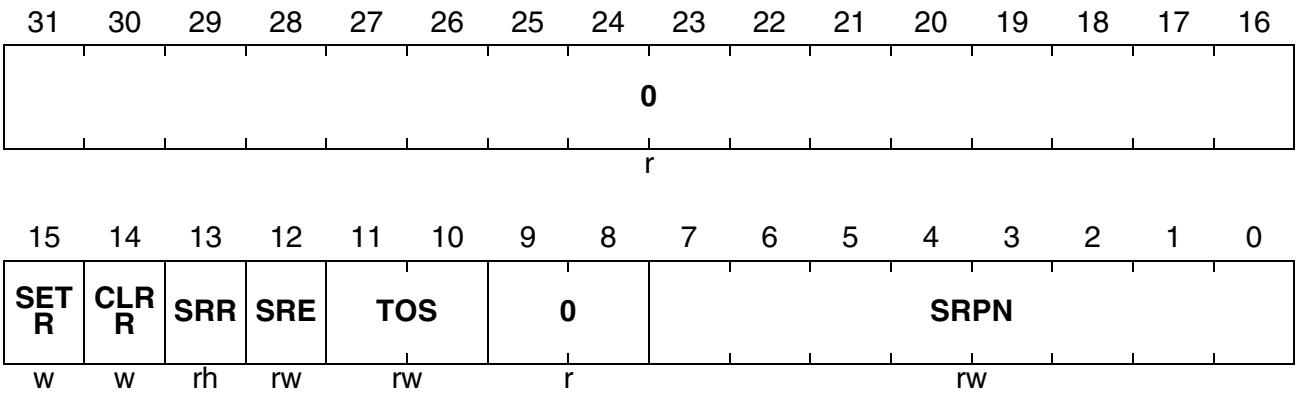


Field	Bits	Type	Description
EVTA	[2:1]	rw	**Event Associated** Specifies the action associated with the event: 000 None; disabled 001 Assert external pin $\overline{\text{BRKOUT}}$ 010 Halt 011 Breakpoint trap 100 Software breakpoint 0 101 Reserved, same behavior as 000 110 Reserved, same behavior as 000 111 Reserved, same behavior as 000
BBM	3	rw	**Break Before Make or Break After Make Selection** 0 Break after make 1 Break before make
SUSP	5	rw	**OCDS Suspend Signal State** Value to be assigned to the OCDS suspend signal when the event is raised.
DLR_LR	8	rw	Controls combination of D_{LR} and C_{LR}
DLR_U	9	rw	Controls combination of D_{LR} and C_U
DU_LR	10	rw	Controls combination of D_U and C_{LR}
DU_U	11	rw	Controls combination of D_U and C_U
0	4, [7:6], [31:12]	r	**Reserved**; read as 0; should be written with 0.

The software breakpoint is controlled by the Software Breakpoint Service Request Control Register 0.

SBSRC0
Software Breakpoint Service Request Control Register 0

Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number** 00_H Software breakpoint service request is never serviced. 01_H Software breakpoint service request is on lowest priority. FF_H Software breakpoint service request is on highest priority.
TOS	[11:10]	rw	**Type of Service Control** 00 CPU service is initiated. 01 Reserved. 1X Reserved.
SRE	12	rw	**Service Request Enable** 0 Software breakpoint service request is disabled. 1 Software breakpoint service request is enabled.
SRR	13	rh	**Service Request Flag** 0 No software breakpoint service request is pending. 1 A software breakpoint service request is pending.

Field	Bits	Type	Description
CLRR	14	w	**Request Clear Bit** CLRR is required to reset SRR. 0 No action 1 Clear SRR; bit value is not stored; read always returns 0; no action if SETR is set too
SETR	15	w	**Request Set Bit** SETR is required to set SRR. 0 No action 1 Set SRR; bit value is not stored; read always returns 0; no action if SETR is set too
0	[9:8], [31:16]	r	**Reserved**; returns 0 if read; should be written with 0.

*Note: Further details on interrupt handling and processing are described in **Chapter 14** of this User's Manual*

19.2 Trace Module

This chapter describes the PC trace support implemented in the TC1765.

19.2.1 Overview

A low level on pin $\overline{\text{OCDSE}}$ during power-on reset ($\overline{\text{PORST}}$ = 0) enables the On-Chip Debug Support. When the On-Chip Debug Support is enabled, the Trace Port of the TC1765T is enabled. Every cycle of the system clock (f_{SYS}), 16 bits of information are sent out via the Trace Port TP[15:0]. These 16 bits indicate the current state of the CPU core. They include the following 3 groups:

- 5 bits of pipeline status information
- An 8-bit indirect PC bus
- 3 bits of breakpoint qualification information

From this information, an emulator can reconstruct a cycle-by-cycle break down of the execution of the CPU. It should be possible to follow in real-time the current PC facilitating advanced tools such as profilers, coverage analysis tools etc. The information may also be captured and used to reconstruct, off-line, a cycle-accurate disassembly of the code being executed within the CPU.

The following sections describe the 3 groups of signals listed above and how they may be used to reconstruct the real time trace.

19.2.2 Pipeline Status Signals

Each cycle, a 5-bit code is sent out over the status signals. The meaning of this code is shown in **Table 19-2** for every cycle except for the first cycle after an indirect branch, when an indirect address sync code is sent (see **Section 19.2.3.1**).

Table 19-2 Pipeline Status Codes

Status	PC increment	Jump	Indirect	Description	Unique[1)]
00000_B	0	no	no	nop	yes
00001_B	2	no	no	–	yes
00010_B	2	yes	no	–	yes
00011_B	2	yes	yes	–	yes
00100_B	0	yes	yes	trap	yes
00101_B	4	yes	no	–	yes
00110_B	4	yes	no	–	yes
00111_B	4	yes	yes	–	yes
01000_B	0	yes	yes	interrupt	yes

Table 19-2 Pipeline Status Codes (cont'd)

Status	PC increment	Jump	Indirect	Description	Unique[1)]
01001_B	6	no	no	–	yes
01010_B	6	yes	no	–	yes
01011_B	6	yes	yes	–	yes
01100_B	–	–	–	Reserved: overrun sync pattern	no
01101_B	8	no	no	–	yes
01110_B	8	yes	no	–	yes
01111_B	8	yes	yes	–	yes
10000_B				Reserved	no
10001_B	10	no	no	–	no
10010_B	10	yes	no	–	no
10011_B	–	–	–	Reserved: invalid for core 1	no
10100_B	–	–	–	Reserved	no
10101_B	12	no	no	–	no
10110_B	12	yes	no	–	no
10111_B	–	–	–	Reserved: invalid for core 1	no
11000_B	–	–	–	Reserved	no
11001_B	–	–	–	Reserved	no
11010_B	–	–	–	Reserved	no
11011_B	–	–	–	Reserved	no
11100_B	–	–	–	Reserved	no
11101_B	–	–	–	Reserved	no
11110_B	–	–	–	Reserved	no
11111_B	–	–	–	Reserved	no

1) See **Section 19.2.2.1**.

Quick Decoding of the Pipeline Status Codes

The pipeline status code is split up into two fields:

- Bits [4:2] indicate PC increment
- Bits [1:0] indicate code type

The code type field can have the values shown in **Table 19-3**.

Table 19-3 Code Type

Code Type	Increment PC	Jump	Indirect	Description
00_B	no	no	–	Special code, trap, interrupt etc.
01_B	yes	no	–	Sequential code
10_B	yes	yes	no	Relative branch
11_B	yes	yes	yes	Indirect branch

For sequential code, the new value of the PC determined from:

– new_PC = PC + ((PC increment + 1) × 2)

19.2.2.1 Synchronizing with the Status and Indirect Streams

Unless the emulator follows the execution of the core from reset, there needs to be a way for the emulator to synchronize with the information coming out of the chip. This process can be performed in two stages.

1. The emulator would synchronize with the pipeline status stream.
2. The emulator would synchronize with the indirect PC stream so that the first PC could be obtained at the next indirect branch.

Pipeline Status Stream

Many of the most common pipeline status codes are unique and no equivalent indirect sync code exists. The unique codes are identified in the last column of **Table 19-2**. By waiting until it sees one of these unique codes, the emulator can synchronize with the pipeline status stream.

Indirect PC Stream

Once the emulator has synchronized with pipeline status stream, it can wait for the first indirect branch. Provided there has not been an overrun, the emulator will then be able to determine the PC, and using that as a starting point it will be able to reconstruct the trace.

19.2.3 Indirect Addresses

The target address of an indirect branch, interrupt, or trap entry are sent out one byte at a time over a dedicated 8-bit bus.

A FIFO is implemented to de-couple the generation of the indirect addresses by the core from the trickling of the addresses out of the chip. The pipeline status and indirect sync encoding has been designed to support a FIFO of up to 4 entries. However, the implementation may have fewer entries. If the FIFO fills up, an indirect address overrun is signalled through a special status code (01100_B).

19.2.3.1 Indirect Sync

The indirect sync is a 5-bit code sent over the status bus after every indirect branch. This code is used to synchronize the status stream to the indirect addresses being sent over the indirect PC bus.

The sync code is interpreted in the following manner:

Table 19-4 Indirect Sync Format

Bits	Name	Description
4	Overrun	Used to determine whether an overrun occurred. 0 Overrun 1 No overrun
[3:0]	Offset	1100_B means overrun occurred Other combinations: number of cycle before first byte of indirect target address will be seen on indirect PC bus.

Overrun

The overrun case occurs when the FIFO fills up that decouples the generation of indirect addresses within the CPU from the ability to transmit those addresses over the 8-bit indirect PC bus. When this scenario arises, the PC of the indirect jump which causes the overrun is lost. This is communicated to the emulator through the indirect PC overrun code 01100_B.

The emulator will then not be able to reconstruct the trace between the time of the indirect jump which caused the overrun, and the next indirect jump that does not also encounter an overrun condition.

The overrun condition should only occur very rarely in normal code. The most common source of indirect branches is when the jump is associated with a return from a function or trap/interrupt handler (RET or RFE). The context model in the first implementation restrict the execution of back-to-back context restores, such that the worst case would be 3 context restores separated by 3 cycles followed by a number of context restores separated by a minimum of 5 cycles. Each context restore generates an indirect PC.

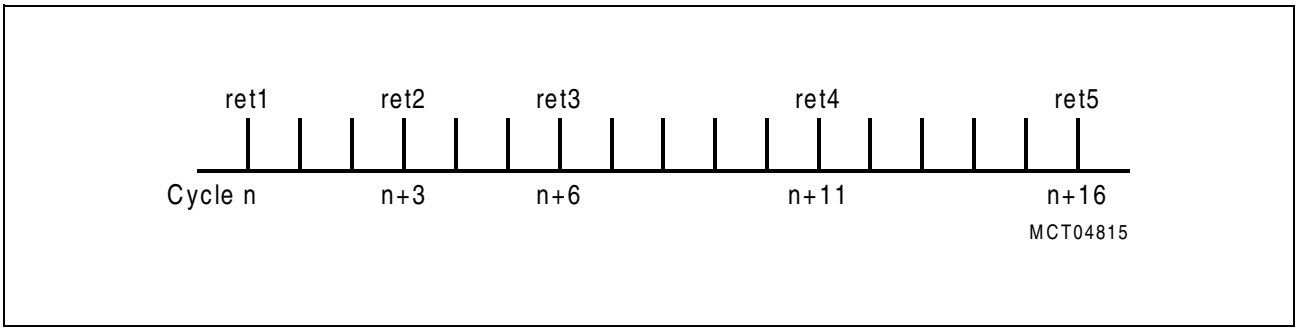


Figure 19-6 Worst Case Back to Back Returns

The FIFO, which is designed to decouple the generation of addresses by the CPU core and the sending out of the indirect PC's over four cycles, can easily handle this scenario. Hence even if the core performs a large number of back to back returns, an overrun would never be generated.

The only scenario that can result in an overrun is several back-to-back jump indirect instructions. This scenario should very seldom be encountered in normal code.

19.2.3.2 Example

```
;a2 contains the address of dest1
;a3 contains the address of dest2
        ji    a2
dest1:  ji    a3

dest2:  add   d9, d8, d7
        ld.w  d5, [a0]24
        ld.w  d6, [a0]28
```

Table 19-5 Trace Example

Cycle	Status					Indirect_pc
	Code	Sync	PC increment	Jump taken	Indirect	
0	–	no	4	yes	yes	X
1	00000_B	yes	–	–	–	t1[7:0]
2	–	no	4	yes	yes	t1[15:8]
3	00010_B	yes	–	–	–	t1[23:16]
4	–	no	8	no	no	t1[31:24]
5	–	no	4	no	no	t2[7:0]

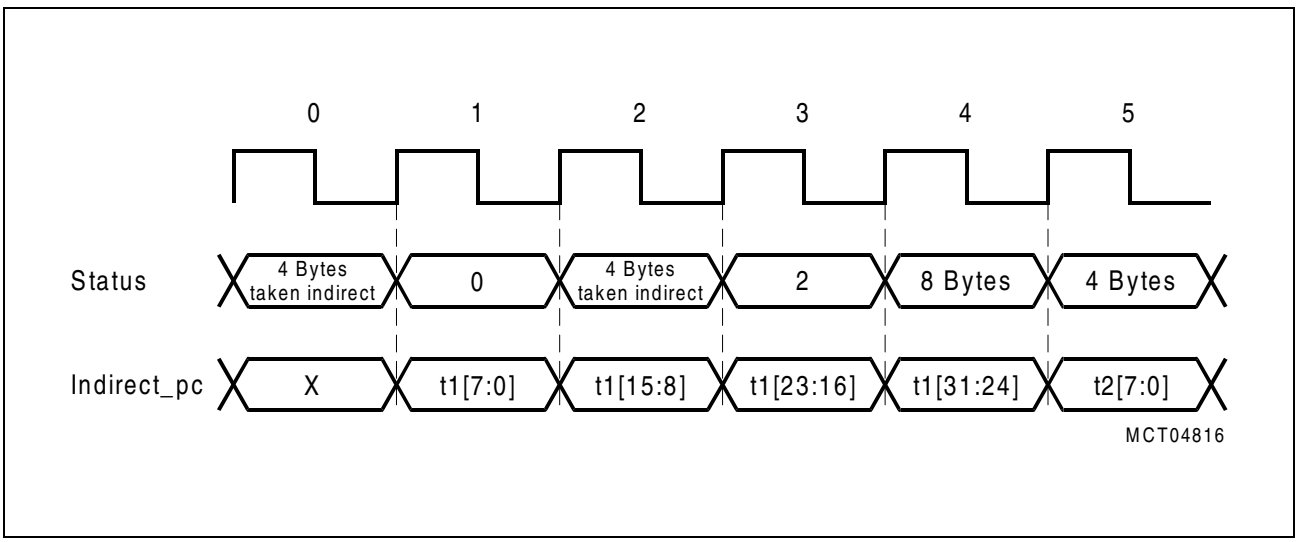


Figure 19-7 Example Output

19.2.4 Trace Output Control (TC1765T only)

This part of the SCU controls the interconnections of the TC1765T Trace Port with the trace interfaces of the Trace Control Unit (TCU) and the and DMA Controller (see also **Chapter 4** in this User's Manual).

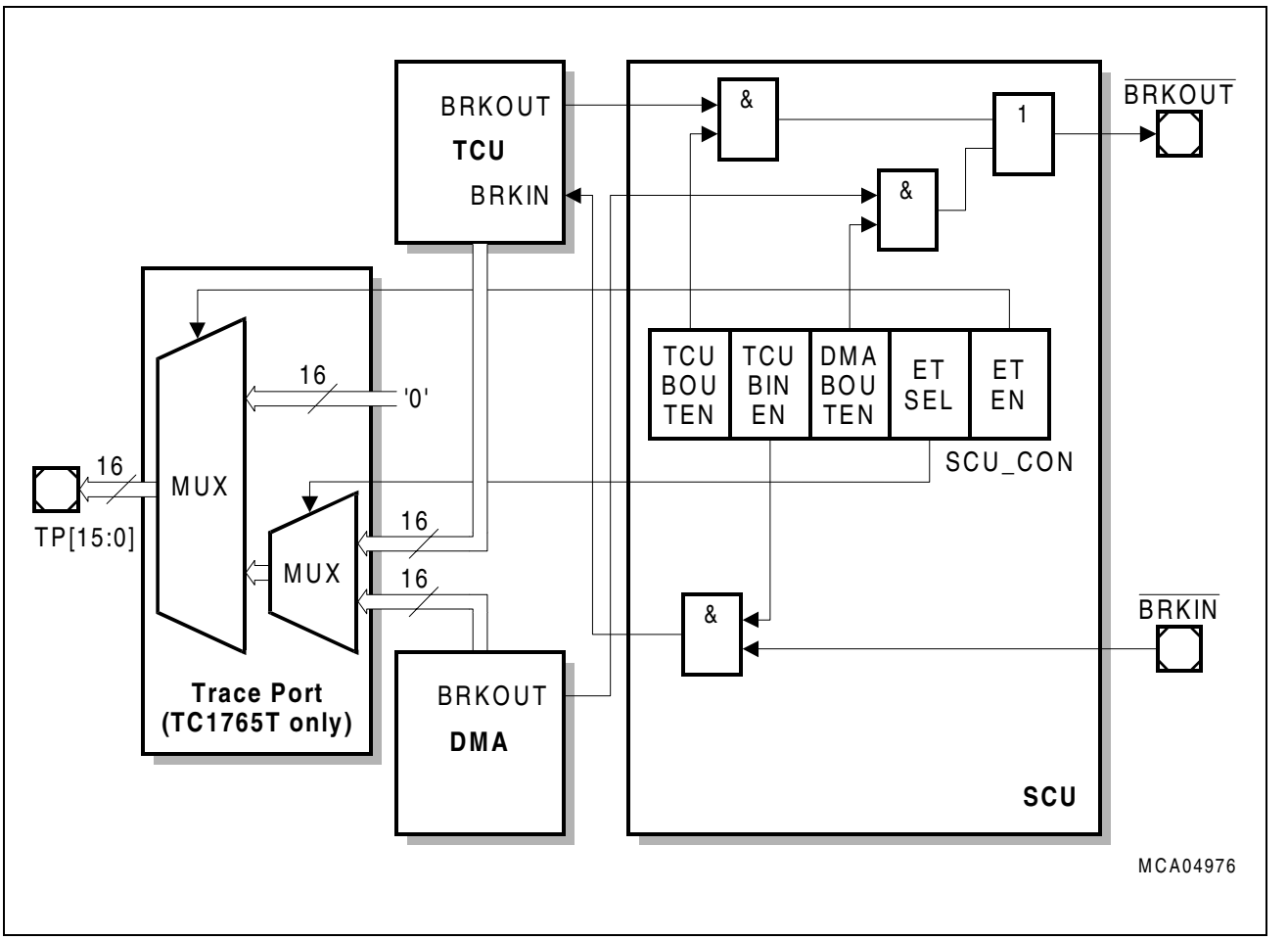


Figure 19-8 Trace Control within the SCU

19.3 DMA Controller Trace

As shown in **Figure 19-1** the DMA Controller is able to output debug information.

Whenever one of the address counters of the DMA is updated a low pulse with the duration of 1 clock cycle t_{SYS} is generated on the $\overline{\text{BRKOUT}}$ signal. (updating of the address counter also means to re-write the same value in case of no pointer modification).

For each of the 16 address counter of the DMA controller one line of the trace port is assigned to indicate which address counter has been updated.

A one at the trace output indicates the address counter has been updated, a zero, that it was inactive.

Table 19-6 Assignment of Trace Output Lines to DMA Channels

Trace Pin	DMA Unit Address Counter
0	Channel 00 Source Address Counter
1	Channel 00 Destination Address Counter
2	Channel 01 Source Address Counter
3	Channel 01 Destination Address Counter
4	Channel 02 Source Address Counter
5	Channel 02 Destination Address Counter
6	Channel 03 Source Address Counter
7	Channel 03 Destination Address Counter
8	Channel 10 Source Address Counter
9	Channel 10 Destination Address Counter
10	Channel 11 Source Address Counter
11	Channel 11 Destination Address Counter
12	Channel 12 Source Address Counter
13	Channel 12 Destination Address Counter
14	Channel 13 Source Address Counter
15	Channel 13 Destination Address Counter

19.4 Debugger Interface (Cerberus)

The Cerberus debug port is provided to debug and emulation tool vendors. The external debug hardware can access the OCDS registers and arbitrary memory locations across the FPI Bus (see **Figure 19-1**). The interface is based on the JTAG standard, and uses only the dedicated JTAG port pins.

The description in this section gives a rough overview for the system programmer of the TC1765 on the operations the debugger interface can perform in the system (and thus effect system behavior). The Cerberus should not be used by an application software since this will disturb the tool behavior. It also describes how the Cerberus is disabled to ensure security in the final product. The information of this section is not sufficient to design tools for the TC1765. For tool developers detailed specifications of Cerberus and the JTAG IO client are necessary.

Features

- External host uses the JTAG pins only
- Allows to address the complete FPI address space
- Performance optimized (protocol)
- Does not use any user resources
- Minimum run time impact
- Generic memory read/write functionality
- Writes words, half words and bytes
- Block read and write support
- Full support for communication between monitor and external debugger
- Supports OCDS for several CPUs on the same FPI Bus
- Pending reads (writes) can be optional triggered from the OCDS module (tracing)

Applications

- Download of programs and data
- Control of the OCDS blocks
- Data acquisition

Performance

The maximum JTAG port clock frequency is 20 MHz. The following performance figures can be achieved:

Table 19-7 Cerberus Performance (Net Data Rates)

Operation	JTAG Clock 200 kHz	JTAG Clock 10 MHz	JTAG Clock 20 MHz
Random read	48 kBit/s	2.4 MBit/s	4.6 MBit/s
Random write	50 kBit/s	2.5 MBit/s	4.9 MBit/s
Block read	104 kBit/s	5.2 MBit/s	10.0 MBit/s
Block write	114 kBit/s	5.7 MBit/s	11.2 MBit/s

19.4.1 RW Mode

The RW mode is used to read or write memory locations by a JTAG host via the JTAG interface. The RW mode needs the FPI Bus master interface of the Cerberus to actively request data reads or writes.

19.4.1.1 Entering RW Mode

RW mode is entered when bit IOSR.RWDIS is 0 and the JTAG host writes a 1 to bit IOCONF.MODE.

19.4.1.2 Data Type Support

The default data type is a 32-bit word. It is used for single word transfers and block transfers. For reading 16-bit registers without getting an FPI Bus error, the IO_READ_HWORD JTAG instruction is provided. If the JTAG host wants to read a byte, it has to read the associated word or half word. In all cases, the read value is 32-bit value and the JTAG host has to extract the needed part by itself.

Writes to bytes or half words are supported with the IO_WRITE_BYTE and IO_WRITE_HWORD JTAG instructions. With this instructions the JTAG host must also shift in the full 32-bit word, but only the selected byte or half word is actually written. Its position is defined by the lowest 2 (bytes) or the second (half word) address bit in IOADDR.

19.4.1.3 FPI Bus Master Interface

The FPI Bus master interface executes the actual read or write of memory locations. It is configured by the IOCONF register and the transactions are requested by the JTAG shift core.

FPI Bus Master Priority Control

There are two different requirements for the RW mode access priority:

- The Cerberus is used to configure the OCDS registers in a CPU. Under these conditions, the Cerberus must be able to set these registers immediately.
- The RW mode is used to read registers while a user program is running.

Under these conditions it is important to influence the real time behavior as little as possible. To allow both options, the FPI Bus master priority can be configured with the FPIPRIO bit in the IOCONF register.

FPI Bus Supervisor Mode

For full debugging support, the external debugger needs the option to access memory locations which are only accessible in supervisor mode. This can be configured with bit IOCONF.SVMODE.

Split Transactions Support

The Cerberus FPI Bus master does not support the full split transactions functionality, but it is able for reads to cooperate with slave peripherals that send a split acknowledge code. In this case, the master interface must wait until the data is sent from the slave.

19.4.2 Communication Mode

In the communication mode the Cerberus has no access to the FPI Bus, But a communication between an JTAG host and a software monitor, which is embedded in the application program, can be established via the Cerberus registers.

The communication mode is the default mode after reset. If the Cerberus is in RW mode, the communication mode is entered when the JTAG host writes a 0 into bit IOCONF.MODE.

19.4.3 System Security

After power-on reset, the Cerberus is in communication mode and needs at least 54 t_{CK} clock cycles to be set into RW mode (20 to set IOPATH and 34 to set IOCONF). If the user program running on the CPU sets bit IOSR.RWDIS immediately after reset, there is no way anymore from outside to set the Cerberus into RW mode via the JTAG interface.

19.4.4 Triggered Transfers

Triggered transfers are an OCDS specific feature of the Cerberus. They can be used to read from or write to a certain memory location, when an OCDS trigger becomes active. Triggered transfers are executed when:

- the Cerberus is in RW mode,
- the bit IOCONF.TRGEN is 1,
- the JTAG shift core has requested a transaction,
- and a positive edge occurs on the *transfer_trigger* signal.

Triggered transfers behave like normal transfers, except that there must be additionally a positive edge on the *transfer_trigger* signal after the JTAG shift core requests the transfer. Another exception is that in case of IO_READ_WORD, IO_READ_HWORD, and IO_READ_BLOCK JTAG instructions the read data is followed by a dirty bit.

19.4.4.1 Tracing of Memory Locations

The main application for triggered transfers is to trace a certain memory location. If a certain memory location is written by a user program, the OCDS module activates a trigger signal. Which trigger signal is selected is defined by the content of bit field IOCONF.CHANNEL. The Cerberus is configured to read the memory location on the

occurrence of this trigger signal. The maximum transfer rate that can be reached is defined by:

$$N_{INSTR} = \frac{46}{T_{INSTR} \times f_{JTAG}}$$

N_{INSTR} is the number of instruction cycles that need to be between two CPU accesses to the memory location. T_{INSTR} is the instruction cycle time of the CPU and f_{JTAG} is the clock rate of the JTAG interface (t_{CK}). For example, if T_{INSTR} = 25 ns and f_{JTAG} = 10 MHz, accesses in every 184th instruction cycle can be fully traced. In many cases this will be sufficient to trace for instance the task ID register. The factor 46 is the sum of 32 for the data, 10 for the JTAG state machine, I/O instruction and start bit and 4 for the synchronization between *transfer_trigger* events and the shift out.

It is recommended that triggered transfers are done with the highest FPI Bus master priority and IOCONF.SVMODE = 1, because otherwise another higher priority master could change the desired data value before it is actually read.

19.4.5 Power Saving

The Cerberus is in power saving mode when it is not selected from the JTAG side. The only register that is always accessible and working is IOSR.

19.4.6 Registers

Table 19-8 Register Summary

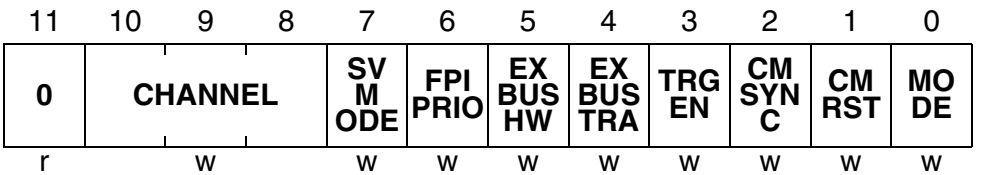
Register	Address	Description
IOCONF	Accessible via JTAG only	Configuration register
IOADDR	Accessible via JTAG only	Address for next RW mode accesses
IODATA	Accessible via JTAG only	RW mode data register
COMDATA	F000 0468$_H$	Communication mode data register
IOSR	F000 046C$_H$	Status register
JDP_ID	F000 0408$_H$	Module ID and revision
TRADDR	Accessible via JTAG only	External bus trace mode address

Note: The Cerberus has fixed absolute register addresses. This is required to make the debug monitor independent of a TriCore product.

19.4.6.1 IOCONF Register

The IOCONF register is used to configure the Cerberus.The IOCONF register is a write-only register for the JTAG host and not accessible from the FPI Bus side.

IOCONF
Cerberus I/O Configuration Register **Reset Value: 0000_H**



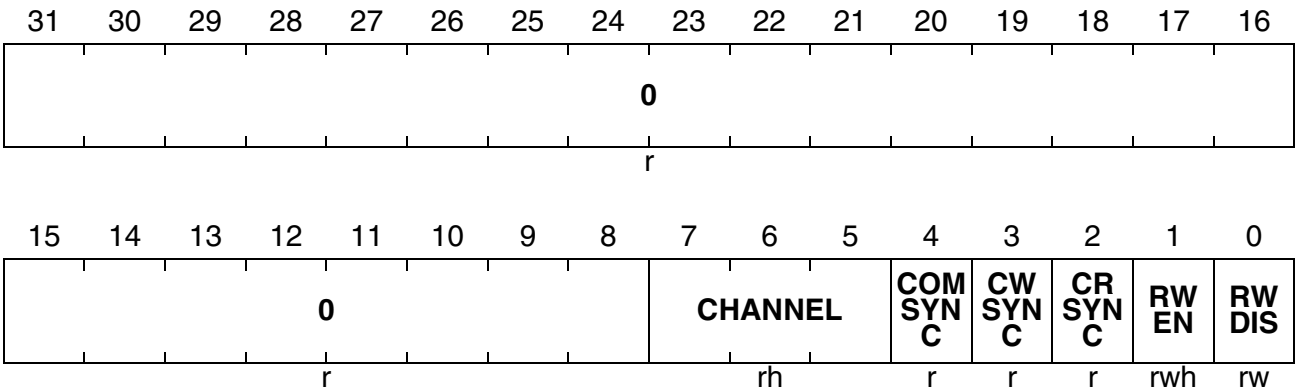
Field	Bits	Type	Description
MODE	0	w	**Mode Selection** This bit defines whether the Cerberus is in RW mode or in communication mode. 0 Communication mode selected 1 RW mode selected
CMRST	1	w	**Communication Mode Bit Reset** This bit is provided to reset the CWSYNC and HWSYNC bits in register IOSR to abort requests in communication mode. This reset is not static, it is only done once when the IOCONF register is updated. 0 Bits CWSYNC and HWSYNC are not affected 1 Bits CWSYNC and HWSYNC in register IOSR are reset
CMSYNC	2	w	**Communication Mode Bit Set** This bit sets the bit COMSYNC in register IOSR. 0 Bit COMSYNC is not affected 1 Bit COMSYNC is set
TRGEN	3	w	**Triggered Transfer Enable** This bit enables triggered transfers in RW mode. 0 Triggered transfers in RW mode are disabled 1 The next RW mode transfers must be triggered by the *transfer_trigger* signal
EXBUSTRA	4	w	**Enable Triggered Transfers to External Bus Address** This bit enables triggered transfers to an external bus address. 0 Trace with external bus address disabled 1 Trace with external bus address enabled

Field	Bits	Type	Description
EXBUSHW	5	w	**FPI Bus Read Size Selection** This bit distinguishes between FPI Bus word and half word reads of the trace source for the external bus trace. 0 The trace source is read with an FPI Bus word access 1 The trace source is read with an FPI Bus half word access
FPIPRIO	6	w	**FPI Bus Master Priority of the Cerberus** This bit sets the priority of the Cerberus FPI Bus master interface in RW mode. 0 Next FPI Bus master request is done with lowest priority 1 Next FPI Bus master request is done with highest priority
SVMODE	7	w	**Supervisor Mode Selection** This bit sets the supervisor mode for the FPI Bus master interface in RW mode. 0 The next RW transfers are not done in supervisor mode 1 The next RW transfers are done in FPI Bus supervisor mode
CHANNEL	[10:8]	w	**Transfer Trigger Selection** This bit field sets the associated bit field in register IOSR and selects the *transfer_trigger[7:0]* signal.
0	[15:11]	r	**Reserved**; should be written with 0.

19.4.6.2 IOSR Register

The IOSR register is used in communication mode and to disable the Cerberus from the CPU side. The IOSR register is only accessible from the FPI Bus side.

IOSR
Cerberus Status Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
RWDIS	0	rw	**Enable Entering RW Mode** This bit is used to prevent the Cerberus from entering RW mode. It can only be set by the CPU in communication mode with an FPI Bus supervisor write operation. If the Cerberus has already entered RW mode, all attempts by the CPU to set this bit are ignored. 0 Entering RW mode is enabled 1 Entering RW mode is disabled This bit is write only in communication mode and in supervisor mode.
RWEN	1	rwh	**User Flag** This bit has no effect on the Cerberus behavior. It is provided for the user program to store whether it has RW mode enabled already or not. RWEN is reset by a JTAG reset only and not by an FPI Bus reset.
CRSYNC	2	r	**Read Synchronization Bit in Communication Mode**
CWSYNC	3	r	**Write Synchronization Bit in Communication Mode**
COMSYNC	4	r	**High Level Communication Synchronization Bit in Communication Code**

Field	Bits	Type	Description
CHANNEL	[7:5]	rh	The CHANNEL bit field is set as COMSYNC by the associated field in the IOCONF register. It is provided to define the CPU (monitor) which is addressed for the next transfer in communication mode on a multi CPU chip.
0	[31:8]	r	**Reserved**; read as 0; should be written with 0.

19.4.6.3 JDP_ID Register

This read only identification register JPD_ID provides the module type and revision number.

19.4.6.4 TRADDR Register

The TRADDR register is used for tracing with external bus address. It defines the uppermost 8 bits of the external bus address. It is set with the IO_SET_TRADDR instruction by the external JTAG host.

19.4.6.5 IOADDR, COMDATA and RWDATA Registers

These registers are used as address, data, and control registers in communication and RW mode.

19.5 Concurrent Debugging

Limited concurrent debugging is possible for CPU and DMA. It is not possible to trace CPU and DMA at the same time, but when the trace port is assigned to the DMA Controller, the break-in and break-out features of the CPU can be used at the same time.

Selecting DMA trace and enabling the break-out lines of both units, allows to distinguish between DMA trace events and CPU breakout activation. During DMA trace the $\overline{\text{BRKOUT}}$ line is activated only together with one of the trace pins being active (‘1’) as well. If $\overline{\text{BRKOUT}}$ is activated with all trace pins being zero, this means that the CPU has activated its break-out signal.

19.6 OCDS Register Address Ranges

In the TC1765, the registers for on-chip OCDS control are located at the following address ranges:

- OCDS Registers except SBSRC0:
  Module Base Address: FFFF FD00$_H$
  Module End Address: FFFF FDFF$_H$
- SBSRC0:
  Module Base Address: FFFF FF00$_H$
  Module End Address: FFFF FFFF$_H$
- Cerberus Registers:
  Module Base Address: F000 0400$_H$
  Module End Address: F000 04FF$_H$
- Absolute Register Address = Module Base Address + Offset Address
  (offset addresses see **Table 19-1**)

20 Register Overview

This chapter defines all registers of the TC1765 and provides the complete address range as well. It also defines the read/write access rights of the specific address ranges/ registers.

Throughout the tables in this chapter, the “Access Mode” “Read” and “Write”, and “Reset Values” columns indicate access rights and values using symbols listed in **Section 20-1**.

Table 20-1 Address Map Symbols

Symbol	Description
U	Access permitted in User Mode 1 only.
SV	Access permitted in Supervisor Mode.
R	Read-only register.
32	Only 32-bit word accesses are permitted to that register/address range.
E	Endinit protected register/address.
PW	Password protected register/address.
NC	No change, indicated register is not changed.
BE	Indicates that an access to this address range generates a Bus Error.
nBE	Indicates that no Bus Error is generated when accessing this address range, even though it is either an access to an undefined address or the access does not follow the given rules.
nE	Indicates that no Error is generated when accessing this address or address range, even though the access is to an undefined address or address range. True for CPU accesses (MTCR/MFCR) to undefined addresses in the CSFR range.
X	Undefined value or bit.

20.1 Segments 0 - 14

20.1.1 Address Map

Table 20-2 shows the block address map of Segment 0 to 14. Special Function Registers are located in Segments 12 and 13 (see Table 20-3).

Note: Bold entries in column "Access Mode" are links to register definitions of the corresponding functional unit.

Table 20-2 Block Address Map of Segments 0 to 14

Segment	Description	Address Range	Access Mode		Size
			Read	Write	
0-7	Reserved	0000 0000$_H$ - 7FFF FFFF$_H$	BE	BE	2 GBytes
8	Reserved	800C 0000$_H$ - 8FFF FFFF$_H$	nE	nE	256 MBytes
9	Reserved	9000 0000$_H$ - 9FFF FFFF$_H$	nE	nE	256 MBytes
10	Reserved for External Memory, cached	A000 0000$_H$ - AFFF FFFF$_H$	nE	nE	256 MBytes
11	Reserved for External Memory, non-cached, mappable into Segment 10	B000 0000$_H$ - BDFF FFFF$_H$	nE	nE	224 MBytes
	External Emulator Space, non-cached	BE00 0000$_H$ - BEFF FFFF$_H$	nE	nE	16 MBytes
	Reserved	BF00 0000$_H$ - BFFF EFFF$_H$	nE	nE	16 MBytes
	8-kByte Boot ROM	BFFF E000$_H$ - BFFF FFFF$_H$	nE	nE	
12	Local Scratch-Pad Code Memory, non-cached (SPRAM)	C000 0000$_H$ - C000 7FFF$_H$	nE	nE	1 KByte[1)]
	Reserved	C000 8000$_H$ - C7FF FEFF$_H$	nE	nE	[1)]
	PMU Registers	C7FF FF00$_H$ - C7FF FFFF$_H$	see Page 20-4		256 Byte
	Reserved	C80C 0000$_H$ - CFFF FFFF$_H$	nE	nE	–

Table 20-2 Block Address Map of Segments 0 to 14 (cont'd)

Segment	Description	Address Range	Access Mode		Size
			Read	Write	
13	Local Data Memory (SRAM) non-cached	D000 0000$_H$ - D000 7FFF$_H$	1)2)		32 KBytes
	Reserved	D000 8000$_H$ - D7FF FEFF$_H$	BE	BE	–
	DMU Registers	D7FF FF00$_H$ - D7FF FFFF$_H$	see Page 20-4)		256 Bytes
	Reserved	D800 0000$_H$ - DFFF FFFF$_H$	BE	BE	128 MBytes
14	Reserved for External Peripheral and Data Memory non-cached	E000 0000$_H$ - EFFF FFFF$_H$	nE	nE	256 MBytes

1) CPU Load/Store accesses to this range can be performed in User or Supervisor Mode. Access width can be 8, 16, 32 or 64 bit, with 8-bit data aligned on byte boundaries and all others aligned on half-word (16-bit) boundaries. Misaligned accesses to the data memory by the CPU's load/store unit will not occur as such conditions will already be handled inside the CPU (Unalignment trap, ALN).

2) The read/write accesses from the FPI Bus can be performed in User or Supervisor Mode. Access width can be 16 or 32 bits, with data aligned on its natural boundary. Misaligned accesses will result in a bus error.

20.1.2 Registers

This section defines the memory map and the addresses of the Special Function Registers of PMU (Segment 12) and DMU (Segment 13).

*Note: Addresses listed in column "Address" of **Table 20-3** are word (32-bit) addresses.*

Table 20-3 Special Function Registers located in Segment 12 and 13

Short Name	Description	Absolute Address	Access Mode		Reset Value
			Read	Write	
PMU Register (Segment 12)					
–	Reserved	C7FF FF00$_H$ - C7FF FF04$_H$	BE	BE	–
PMU_ID	PMU Module Identification Register	C7FF FF08$_H$	U, SV	BE	XXXXXXXX$_H$
–	Reserved	C7FF FF0C$_H$	BE	BE	–
PMU_CON	PMU Control Register	C7FF FF10$_H$	U, SV	SV, 32	0400 3F06$_H$
–	Reserved	C7FF FF14$_H$	BE	BE	–
PMU_ EIFCON	PMU External Instruction Fetch Control Register	C7FF FF18$_H$	U, SV	SV, 32	0000 001F$_H$
–	Reserved	C7FF FF1C$_H$ - C7FF FFFC$_H$	BE	BE	–
DMU Register (Segment 13)					
–	Reserved	D7FF FF00$_H$	nBE	nBE	–
–	Reserved	D7FF FF04$_H$	BE	BE	3)
DMU_ID	DMU Module Identification Register	D7FF FF08$_H$	U, SV	BE	XXXXXXXX$_H$
–	Reserved	D7FF FF0C$_H$	BE	BE	3)
DMU_CON	DMU Control Register	D7FF FF10$_H$	U, SV	SV	0000 0000$_H$
–	Reserved	D7FF FF14$_H$	BE	BE	3)
DMU_STR	DMU Synchronous Trap Flag Register	D7FF FF18$_H$	U, SV 2)	BE 1)	0000 0000$_H$
–	Reserved	D7FF FF1C$_H$	BE	BE	3)
DMU_ATR	DMU Asynchronous Trap Flag Register	D7FF FF20$_H$	U, SV 2)	BE 1)	0000 0000$_H$
–	Reserved	D7FF FF24$_H$	BE	BE	3)
–	Reserved	D7FF FF28$_H$	nBE	nBE	–

Table 20-3 Special Function Registers located in Segment 12 and 13 (cont'd)

Short Name	Description	Absolute Address	Access Mode		Reset Value
			Read	Write	
–	Reserved	D7FF FF2C$_H$	BE	BE	3)
–	Reserved	D7FF FF30$_H$	nBE	nBE	–
–	Reserved	D7FF FF34$_H$	BE	BE	3)
–	Reserved	D7FF FF38$_H$	nBE	nBE	–
–	Reserved	D7FF FF3C$_H$	BE	BE	3)
–	Reserved	D7FF FF40$_H$	nBE	nBE	–
–	Reserved	D7FF FF44$_H$	BE	BE	3)
–	Reserved	D7FF FF48$_H$	nBE	nBE	–
–	Reserved	D7FF FF4C$_H$	BE	BE	3)
–	Reserved	D7FF FF50$_H$	nBE	nBE	–
–	Reserved	D7FF FF54$_H$	BE	BE	3)
–	Reserved	D7FF FF58$_H$	nBE	nBE	–
–	Reserved	D7FF FF5C$_H$	BE	BE	3)
–	Reserved	D7FF FF60$_H$	nBE	nBE	–
–	Reserved	D7FF FF64$_H$	BE	BE	3)
–	Reserved	D7FF FF68$_H$	nBE	nBE	–
–	Reserved	D7FF FF6C$_H$	BE	BE	3)
–	Reserved	D7FF FF70$_H$	nBE	nBE	–
–	Reserved	D7FF FF74$_H$	BE	BE	3)
–	Reserved	D7FF FF78$_H$	nBE	nBE	–
–	Reserved	D7FF FF7C$_H$	BE	BE	3)
DMU_ IOCR0	DMU Internal Overlay Control Register 0	D7FF FF80$_H$	U, SV 1)	SV 1)	0000 0000$_H$
–	Reserved	D7FF FF84$_H$	BE	BE	3)
DMU_ IOCR1	DMU Internal Overlay Control Register 1	D7FF FF88$_H$	U, SV 1)	SV 1)	0000 0000$_H$
–	Reserved	D7FF FF8C$_H$	BE	BE	3)
DMU_ IOCR2	DMU Internal Overlay Control Register 2	D7FF FF90$_H$	U, SV 1)	SV 1)	0000 0000$_H$
–	Reserved	D7FF FF94$_H$	BE	BE	3)

Table 20-3 Special Function Registers located in Segment 12 and 13 (cont'd)

Short Name	Description	Absolute Address	Access Mode		Reset Value
			Read	Write	
DMU_ IOCR3	DMU Internal Overlay Control Register 3	D7FF FF98$_H$	U, SV 1)	SV 1)	0000 0000$_H$
–	Reserved	D7FF FF9C$_H$	BE	BE	3)
DMU_ EOCR0	DMU External Overlay Control Register 0	D7FF FFA0$_H$	U, SV 1)	SV 1)	0000 0000$_H$
–	Reserved	D7FF FFA4$_H$	BE	BE	3)
DMU_ EOCR1	DMU External Overlay Control Register 1	D7FF FFA8$_H$	U, SV 1)	SV 1)	0000 0000$_H$
–	Reserved	D7FF FFAC$_H$	BE	BE	3)
–	Reserved	D7FF FFB0$_H$	nBE	nBE	–
–	Reserved	D7FF FFB4$_H$	BE	BE	3)
DMU_ IORBAP	DMU Internal Overlay RAM Base Address Page Register	D7FF FFB8$_H$	U, SV 1)	SV 1)	0000 0000$_H$
–	Reserved	D7FF FFBC$_H$	BE	BE	3)
–	Reserved	D7FF FFC0$_H$	nBE	nBE	–
–	Reserved	D7FF FFC4$_H$	BE	BE	3)
–	Reserved	D7FF FFC8$_H$	nBE	nBE	–
–	Reserved	D7FF FFCC$_H$	BE	BE	3)
–	Reserved	D7FF FFD0$_H$	nBE	nBE	–
–	Reserved	D7FF FFD4$_H$	BE	BE	3)
–	Reserved	D7FF FFD8$_H$	nBE	nBE	–
–	Reserved	D7FF FFDC$_H$	BE	BE	3)
–	Reserved	D7FF FFE0$_H$	nBE	nBE	–
–	Reserved	D7FF FFE4$_H$	BE	BE	3)
–	Reserved	D7FF FFE8$_H$	nBE	nBE	–
–	Reserved	D7FF FFEC$_H$	BE	BE	3)
–	Reserved	D7FF FFF0$_H$	nBE	nBE	–
–	Reserved	D7FF FFF4$_H$	BE	BE	3)

Table 20-3 Special Function Registers located in Segment 12 and 13 (cont'd)

Short Name	Description	Absolute Address	Access Mode		Reset Value
			Read	Write	
–	Reserved	D7FF FFF8$_H$	nBE	nBE	–
–	Reserved	D7FF FFFC$_H$	BE	BE	3)

1) Access to the DMU registers must only be made with double-word-aligned word accesses. An access not conforming to this rule, or an access which does not follow the specified privilege mode (supervisor mode), or a write access to a read-only register, will cause a bus error if the access was from the FPI Bus, or to a trap, flagged with a DMU Control Register Error Flag (see DMUSTR/DMUATR registers) in case of a CPU load/ store access.

2) Reading this register in supervisor mode returns the contents and then clears the register. Reading it in user mode only returns the contents of the register and does not clear its bits. No error will be reported in this case.

3) A read access to this range will lead to a bus error on data load operation trap. A write access to this range will lead to a bus error on data store operation trap.

20.2 Segment 15 (Peripheral Units)

20.2.1 Address Map

Table 20-4 and Table 20-5 show the memory map and registers of Segment 15.

Note: Bold entries in column "Access Mode" are links to the register definitions of the corresponding functional unit.

Table 20-4 Block Address Map of Segment 15

Unit	Address Range	Access Mode		Size
		Read	Write	
System Control Unit (SCU) and Watchdog Timer (WDT)	$F000\ 0000_H$ - $F000\ 00FF_H$	see Page 20-11		256 Bytes
Reserved	$F000\ 0100_H$ - $F000\ 01FF_H$	BE	BE	–
Bus Control Unit (BCU)	$F000\ 0200_H$ - $F000\ 02FF_H$	see Page 20-12		256 Bytes
System Timer (STM)	$F000\ 0300_H$ - $F000\ 03FF_H$	see Page 20-13		256 Bytes
On-Chip Debug Support (Cerberus)	$F000\ 0400_H$ - $F000\ 04FF_H$	see Page 20-13		256 Bytes
External Bus Unit (EBU)	$F000\ 0500_H$ - $F000\ 05FF_H$	see Page 20-14		256 Bytes
Reserved	$F000\ 0600_H$ - $F000\ 06FF_H$	BE	BE	–
General Purpose Timer Unit (GPTU)	$F000\ 0700_H$ - $F000\ 07FF_H$	see Page 20-15		256 Bytes
Asynchronous/Synchronous Serial Interface 0 (ASC0)	$F000\ 0800_H$ - $F000\ 08FF_H$	see Page 20-17		256 Bytes
Asynchronous/Synchronous Serial Interface 1 (ASC1)	$F000\ 0900_H$ - $F000\ 09FF_H$	see Page 20-18		256 Bytes
High-Speed Synchronous Serial Interface 0 (SSC0)	$F000\ 0A00_H$ - $F000\ 0AFF_H$	see Page 20-19		256 Bytes
High-Speed Synchronous Serial Interface 1 (SSC1)	$F000\ 0B00_H$ - $F000\ 0BFF_H$	see Page 20-20		256 Bytes
Reserved	$F000\ 0C00_H$ - $F000\ 17FF_H$	BE	BE	–

Table 20-4 Block Address Map of Segment 15 (cont'd)

Unit	Address Range	Access Mode		Size
		Read	Write	
General Purpose Timer Array (GPTA)[1)]	$F000\ 1800_H$ - $F000\ 1FFF_H$	see Page 20-21		8 × 256 Bytes
Reserved	$F000\ 2000_H$ - $F000\ 21FF_H$	BE	BE	–
Analog to Digital Converter 0 (ADC0)[1)]	$F000\ 2200_H$ - $F000\ 23FF_H$	see Page 20-41		2 × 256 Bytes
Analog to Digital Converter 1 (ADC1)[1)]	$F000\ 2400_H$ - $F000\ 25FF_H$	see Page 20-46		2 × 256 Bytes
Reserved	$F000\ 2600_H$ - $F000\ 27FF_H$	BE	BE	–
Port 0	$F000\ 2800_H$ - $F000\ 28FF_H$	see Page 20-51		256 Bytes
Port 1	$F000\ 2900_H$ - $F000\ 29FF_H$	see Page 20-52		256 Bytes
Port 2	$F000\ 2A00_H$ - $F000\ 2AFF_H$	see Page 20-53		256 Bytes
Port 3	$F000\ 2B00_H$ - $F000\ 2BFF_H$	see Page 20-54		256 Bytes
Port 4	$F000\ 2C00_H$ - $F000\ 2CFF_H$	see Page 20-54		256 Bytes
Port 5	$F000\ 2D00_H$ - $F000\ 2DFF_H$	see Page 20-55		256 Bytes
Reserved	$F000\ 2E00_H$ - $F000\ 3BFF_H$	BE	BE	–
DMA Controller	$F000\ 3C00_H$ - $F000\ 3DFF_H$	see Page 20-56		2 × 256 Bytes
Reserved	$F000\ 3E00_H$ - $F00F\ FFFF_H$	BE	BE	–
Controller Area Network Module (CAN)[1)]	$F010\ 0000_H$ - $F010\ 0BFF_H$	see Page 20-60		12 × 256 Bytes
Reserved	$F010\ 0C00_H$ - $FFFE\ FEFF_H$	BE	BE	–

Table 20-4 Block Address Map of Segment 15 (cont'd)

Unit		Address Range	Access Mode		Size
			Read	Write	
CPU	CPU Slave Interface Registers (CPS)	FFFE FF00$_H$ - FFFE FFFF$_H$	see Page 20-78		256 Bytes
	Reserved	FFFF 0000$_H$ - FFFF BFFF$_H$	nE	nE	–
	Memory Protection Register	FFFF C000$_H$ - FFFF EFFF$_H$	see Page 20-79		48 × 256 Bytes
	Reserved	FFFF F000$_H$ - FFFF FCFF$_H$	nE	nE	–
	Core Debug Register	FFFF FD00$_H$ - FFFF FDFF$_H$	see Page 20-81		256 Bytes
	Core Special Function Registers (CSFR)	FFFF FE00$_H$ - FFFF FEFF$_H$	see Page 20-82		256 Bytes
	General Purpose Register (GPR)	FFFF FF00$_H$ - FFFF FFFF$_H$	see Page 20-83		256 Bytes

1) Accesses to unused address regions within these peripheral units do not generate a bus errors.

20.2.2 Registers

Table 20-5 shows the address map with all register of Segment 15.

Note: Addresses listed in column "Address" of Table 20-5 are word (32-bit) addresses.

Table 20-5 Detailed Address Map of Segment 15

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
System Control Unit (SCU) with Watchdog Timer (WDT)					
–	Reserved	F000 0000$_H$ - F000 0004$_H$	BE	BE	–
SCU_ID	SCU Module Identification Register	F000 0008$_H$	U, SV	SV	XXXXXXXX$_H$
–	Reserved	F000 000C$_H$	BE	BE	–
RST_REQ	Reset Request Register	F000 0010$_H$	U, SV	U, SV, E	0000 0000$_H$
RST_SR	Reset Status Register	F000 0014$_H$	U, SV	U, SV	according boot cfg.
–	Reserved	F000 0018$_H$ - F000 001C$_H$	BE	BE	–
WDT_CON0	Watchdog Timer Control Register 0	F000 0020$_H$	U, SV	U, SV, PW	FFFC 0002$_H$
WDT_CON1	Watchdog Timer Control Register 1	F000 0024$_H$	U, SV	U, SV, E	0000 0000$_H$
WDT_SR	Watchdog Timer Status Register	F000 0028$_H$	U, SV	U, SV, NC	FFFC 0010$_H$
NMISR	NMI Status Register	F000 002C$_H$	U, SV	U, SV	0000 0000$_H$
PMG_CON	Power Management Control Register	F000 0030$_H$	U, SV	U, SV, E	0000 0001$_H$
PMG_CSR	Power Management Control and Status Reg.	F000 0034$_H$	U, SV	U, SV	0000 0100$_H$
–	Reserved	F000 0038$_H$ - F000 003C$_H$	BE	BE	–
PLL_CLC	PLL Clock Control Reg.	F000 0040$_H$	U, SV	U, SV, E	0007 UU00$_H$
–	Reserved	F000 0044$_H$ - F000 004C$_H$	BE	BE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
SCU_CON	SCU Control Register	$F000\ 0050_H$	U, SV	U, SV	$00F0\ 0030_H$
SCU_ TRSTAT	SCU Trace Status Register	$F000\ 0054_H$	U, SV	U, SV	$0000\ 0000_H$
–	Reserved	$F000\ 0058_H$ - $F000\ 006C_H$	BE	BE	–
MANID	Manufacturer Identification Register	$F000\ 0070_H$	U, SV	U, SV	$0000\ 1820_H$
CHIPID	Chip Identification Reg.	$F000\ 0074_H$	U, SV	U, SV	$0000\ 8601_H$
RTID	Redesign Tracing Identification Register	$F000\ 0078_H$	U, SV	U, SV	$0000\ 0000_H$
–	Reserved	$F000\ 007C_H$ - $F000\ 00FC_H$	BE	BE	–
Bus Control Unit (BCU)					
–	Reserved	$F000\ 0200_H$ - $F000\ 0204_H$	BE	nBE	–
BCU_ID	BCU Module Identification Register	$F000\ 0208_H$	U, SV	U, SV	$XXXXXXXX_H$
–	Reserved	$F000\ 020C_H$	BE	BE	–
BCU_CON	BCU Control Register	$F000\ 0210_H$	U, SV, 32	SV, 32	$4009\ FFFF_H$
–	Reserved	$F000\ 0214_H$ - $F000\ 021C_H$	BE	BE	–
BCU_ECON	BCU Error Control Capture Register	$F000\ 0220_H$	U, SV, 32	SV, 32	$0000\ 0000_H$
BCU_EADD	BCU Error Address Capture Register	$F000\ 0224_H$	U, SV, 32	SV, 32	$0000\ 0000_H$
BCU_EDAT	BCU Error Data Capture Register	$F000\ 0228_H$	U, SV, 32	SV, 32	$0000\ 0000_H$
–	Reserved	$F000\ 022C_H$ - $F000\ 02F8_H$	BE	BE	–
BCU_SRC	BCU Service Request Control Register	$F000\ 02FC_H$	U, SV, 32	SV, 32	$0000\ 0000_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
System Timer (STM)					
STM_CLC	STM Clock Control Reg.	F000 0300$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 0304$_H$	BE	BE	–
STM_ID	STM Module Identification Register	F000 0308$_H$	U, SV	SV	XXXXXXXX$_H$
–	Reserved	F000 030C$_H$	BE	BE	–
STM_TIM0	STM Timer Register 0	F000 0310$_H$	U, SV	SV	0000 0000$_H$
STM_TIM1	STM Timer Register 1	F000 0314$_H$	U, SV	SV	0000 0000$_H$
STM_TIM2	STM Timer Register 2	F000 0318$_H$	U, SV	SV	0000 0000$_H$
STM_TIM3	STM Timer Register 3	F000 031C$_H$	U, SV	SV	0000 0000$_H$
STM_TIM4	STM Timer Register 4	F000 0320$_H$	U, SV	SV	0000 0000$_H$
STM_TIM5	STM Timer Register 5	F000 0324$_H$	U, SV	SV	0000 0000$_H$
STM_TIM6	STM Timer Register 6	F000 0328$_H$	U, SV	SV	0000 0000$_H$
STM_CAP	STM Timer Capture Reg.	F000 032C$_H$	U, SV	SV	0000 0000$_H$
–	Reserved	F000 0330$_H$ - F000 03FC$_H$	BE	BE	–
On-Chip Debug Support					
–	Reserved	F000 0400$_H$ - F000 0404$_H$	BE	BE	–
JPD_ID	JTAG/OCDS Module Identification Register	F000 0408$_H$	U, SV	BE	XXXXXXXX$_H$
–	Reserved	F000 040C$_H$ - F000 0464$_H$	BE	BE	–
COMDATA	Cerberus Communication Mode Data Register	F000 0468$_H$	SV	SV	0000 0000$_H$
IOSR	Cerberus Status Register	F000 046C$_H$	SV	SV	0000 0000$_H$
–	Reserved	F000 0470$_H$ - F000 04FC$_H$	BE	BE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
External Bus Unit (EBU)					
EBU_CLC	EBU Clock Control Reg.	$F000\ 0500_H$	U, SV	U, SV, E	$0000\ 0000_H$
–	Reserved	$F000\ 0504_H$	BE	BE	–
EBU_ID	EBU Module Identification Register	$F000\ 0508_H$	U, SV	SV	$XXXXXXXX_H$
–	Reserved	$F000\ 050C_H$	BE	BE	–
EBU_CON	EBU Global Control Reg.	$F000\ 0510_H$	U, SV	U, SV	$0000\ 0068_H$
–	Reserved; this location must not be written.	$F000\ 0514_H$ - $F000\ 051C_H$	nBE	nBE	–
EBU_ ADDSEL0	EBU Address Select Register 0	$F000\ 0520_H$	U, SV	U, SV	$0000\ 0000_H$ $A000\ 0001_H$
EBU_ ADDSEL1	EBU Address Select Register 1	$F000\ 0524_H$	U, SV	U, SV	$0000\ 0000_H$ $A000\ 0001_H$
EBU_ ADDSEL2	EBU Address Select Register 2	$F000\ 0528_H$	U, SV	U, SV	$0000\ 0000_H$ $A000\ 0001_H$
EBU_ ADDSEL3	EBU Address Select Register 3	$F000\ 052C_H$	U, SV	U, SV	$0000\ 0000_H$ $A000\ 0001_H$
–	Reserved	$F000\ 0530_H$ - $F000\ 055C_H$	BE	BE	–
EBU_ BUSCON0	EBU Bus Configuration Register 0	$F000\ 0560_H$	U, SV	U, SV	$E802\ 61FF_H$
EBU_ BUSCON1	EBU Bus Configuration Register 1	$F000\ 0564_H$	U, SV	U, SV	$E802\ 61FF_H$
EBU_ BUSCON2	EBU Bus Configuration Register 2	$F000\ 0568_H$	U, SV	U, SV	$E802\ 61FF_H$
EBU_ BUSCON3	EBU Bus Configuration Register 3	$F000\ 056C_H$	U, SV	U, SV	$E802\ 61FF_H$
–	Reserved	$F000\ 0570_H$ - $F000\ 057C_H$	BE	BE	–
EBU_ EMUAS	EBU Emulator Address Select Register	$F000\ 0580_H$	U, SV	U, SV	$BE00\ 0031_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
EBU_ EMUBC	EBU Emulator Bus Configuration Register	F000 0584$_H$	U, SV	U, SV	0016 0280$_H$
EBU_ EMUCON	EBU Emulator Configuration Register	F000 0588$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 058C$_H$ - F000 05FC$_H$	BE	BE	–
General Purpose Timer Unit (GPTU)					
GPTU_ CLC	GPTU Clock Control Reg.	F000 0700$_H$	U, SV	U, SV, E	0000 0002$_H$
–	Reserved	F000 0704$_H$	nBE	nBE	–
GPTU_ID	GPTU Module Identification Register	F000 0708$_H$	U, SV	BE	XXXXXXXX$_H$
–	Reserved	F000 070C$_H$	nBE	nBE	–
GPTU_ T01IRS	GPTU Timers T0 and T1 Input and Reload Source Selection Register	F000 0710$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T01OTS	GPTU Timers T0 and T1 Output, Trigger and Service Req. Register	F000 0714$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T2CON	GPTU Timer T2 Control Register	F000 0718$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T2RCCON	GPTU Timer T2 Reload/ Capture Control Register	F000 071C$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T2AIS	GPTU Timer T2/T2A Ext. Input Selection Register	F000 0720$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T2BIS	GPTU Timer T2B External Input Selection Register	F000 0724$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T2ES	GPTU Timer T2 External Input Edge Selection Reg.	F000 0728$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ OSEL	GPTU Output Source Selection Register	F000 072C$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_OUT	GPTU Output Register	F000 0730$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTU_ T0DCBA	GPTU Timer T0 Count Register (T0D, T0C, T0B, T0A)	F000 0734$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T0CBA	GPTU Timer T0 Count Register (T0C, T0B, T0A)	F000 0738$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T0RDCBA	GPTU Timer T0 Reload Register (T0RD, T0RC, T0RB, T0RA)	F000 073C$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T0RCBA	GPTU Timer T0 Reload Register (T0RC, T0RB, T0RA)	F000 0740$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T1DCBA	GPTU Timer T1 Count Register (T1D, T1C, T1B, T1A)	F000 0744$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T1CBA	GPTU Timer T1 Count Register (T1C, T1B, T1A)	F000 0748$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T1RDCBA	GPTU Timer T1 Reload Register (T1RD, T1RC, T1RB, T1RA)	F000 074C$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T1RCBA	GPTU Timer T1 Reload Register (T1RC, T1RB, T1RA)	F000 0750$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_T2	GPTU Timer T2 Count Register	F000 0754$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T2RC0	GPTU Timer T2 Reload/ Capture Register 0	F000 0758$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T2RC1	GPTU Timer T2 Reload/ Capture Register 1	F000 075C$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ T012RUN	GPTU Timers T0, T1, T2 Run Control Register	F000 0760$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0764$_H$ - F000 07D8$_H$	BE	BE	–
GPTU_ SRSEL	GPTU Service Request Source Select Register	F000 07DC$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTU_ SRC7	GPTU Service Request Control Register 7	F000 07E0$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ SRC6	GPTU Service Request Control Register 6	F000 07E4$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ SRC5	GPTU Service Request Control Register 5	F000 07E8$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ SRC4	GPTU Service Request Control Register 4	F000 07EC$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ SRC3	GPTU Service Request Control Register 3	F000 07F0$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ SRC2	GPTU Service Request Control Register 2	F000 07F4$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ SRC1	GPTU Service Request Control Register 1	F000 07F8$_H$	U, SV	U, SV	0000 0000$_H$
GPTU_ SRC0	GPTU Service Request Control Register 0	F000 07FC$_H$	U, SV	U, SV	0000 0000$_H$
Asynchronous/Synchronous Serial Interface 0 (ASC0)					
ASC0_ CLC	ASC0 Clock Control Reg.	F000 0800$_H$	U, SV	U, SV, E	0000 0002$_H$
–	Reserved	F000 0804$_H$	nBE	nBE	–
ASC0_ID	ASC0 Module Identification Register	F000 0808$_H$	U, SV	U, SV	XXXXXXXX$_H$
–	Reserved	F000 080C$_H$	BE	BE	–
ASC0_CON	ASC0 Control Register	F000 0810$_H$	U, SV	U, SV	0000 0000$_H$
ASC0_BG	ASC0 Baud Rate/Timer Reload Register	F000 0814$_H$	U, SV	U, SV	0000 0000$_H$
ASC0_FDV	ASC0 Fractional Divider Register	F000 0818$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 081C$_H$	BE	BE	–
ASC0_ TBUF	ASC0 Transmit Buffer Register	F000 0820$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ASC0_ RBUF	ASC0 Receive Buffer Register	F000 0824$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0828$_H$ - F000 08D4$_H$	BE	BE	–
–	Reserved	F000 08D8$_H$ - F000 08DC$_H$	nBE	nBE	–
–	Reserved	F000 08E0$_H$ - F000 08EC$_H$	BE	BE	–
ASC0_ TSRC	ASC0 Transmit Interrupt Service Req. Control Reg.	F000 08F0$_H$	U, SV	U, SV	0000 0000$_H$
ASC0_ RSRC	ASC0 Receive Interrupt Service Req. Control Reg.	F000 08F4$_H$	U, SV	U, SV	0000 0000$_H$
ASC0_ ESRC	ASC0 Error Interrupt Service Req. Control Reg.	F000 08F8$_H$	U, SV	U, SV	0000 0000$_H$
ASC0_ TBSRC	ASC0 Transmit Buffer Interrupt Service Req. Control Reg.	F000 08FC$_H$	U, SV	U, SV	0000 0000$_H$
Asynchronous/Synchronous Serial Interface 1 (ASC1)					
ASC1_CLC	ASC1 Clock Control Reg.	F000 0900$_H$	U, SV	U, SV, E	0000 0002$_H$
–	Reserved	F000 0904$_H$	nBE	nBE	–
ASC1_ID	ASC1 Module Identification Register	F000 0908$_H$	U, SV	U, SV	XXXXXXXX$_H$
–	Reserved	F000 090C$_H$	BE	BE	–
ASC1_CON	ASC1 Control Register	F000 0910$_H$	U, SV	U, SV	0000 0000$_H$
ASC1_BG	ASC1 Baud Rate/Timer Reload Register	F000 0914$_H$	U, SV	U, SV	0000 0000$_H$
ASC1_FDV	ASC1 Fractional Divider Register	F000 0918$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 091C$_H$	BE	BE	–
ASC1_ TBUF	ASC1 Transmit Buffer Register	F000 0920$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ASC1_ RBUF	ASC1 Receive Buffer Register	F000 0924$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0928$_H$ - F000 09D4$_H$	BE	BE	–
–	Reserved	F000 09D8$_H$ - F000 09DC$_H$	nBE	nBE	–
–	Reserved	F000 09E0$_H$ - F000 09EC$_H$	BE	BE	–
ASC1_ TSRC	ASC1 Transmit Interrupt Service Req. Control Reg.	F000 09F0$_H$	U, SV	U, SV	0000 0000$_H$
ASC1_ RSRC	ASC1 Receive Interrupt Service Req. Control Reg.	F000 09F4$_H$	U, SV	U, SV	0000 0000$_H$
ASC1_ ESRC	ASC1 Error Interrupt Service Req. Control Reg.	F000 09F8$_H$	U, SV	U, SV	0000 0000$_H$
ASC1_ TBSRC	ASC1 Transmit Buffer Interrupt Service Req. Control Reg.	F000 09FC$_H$	U, SV	U, SV	0000 0000$_H$
High-Speed Synchronous Serial Interface 0 (SSC0)					
SSC0_CLC	SSC0 Clock Control Register	F000 0A00$_H$	U, SV	U, SV, E	0000 0002$_H$
–	Reserved	F000 0A04$_H$	nBE	nBE	–
SSC0_ID	SSC0 Module Identification Register	F000 0A08$_H$	U, SV	BE	XXXXXXXX$_H$
–	Reserved	F000 0A0C$_H$	BE	BE	–
SSC0_CON	SSC0 Control Register	F000 0A10$_H$	U, SV	U, SV	0000 0000$_H$
SSC0_BR	SSC0 Baud Rate Timer Reload Register	F000 0A14$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0A18$_H$ - F000 0A1C$_H$	BE	BE	–
SSC0_TB	SSC0 Transmit Buffer Register	F000 0A20$_H$	U, SV	U, SV	0000 0000$_H$
SSC0_RB	SSC0 Receive Buffer Register	F000 0A24$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
–	Reserved	F000 0A28$_H$ - F000 0A2C$_H$	BE	BE	–
SSC0_ RXFCON	SSC0 Receive FIFO Control Register	F000 0A30$_H$	U, SV	U, SV	0000 0100$_H$
SSC0_ TXFCON	SSC0 Transmit FIFO Control Register	F000 0A34$_H$	U, SV	U, SV	0000 0100$_H$
SSC0_ FSTAT	SSC0 FIFO Status Register	F000 0A38$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0A3C$_H$	BE	BE	–
SSC0_ WHBCON	SSC0 Write Hardware Modified Control Register Bits	F000 0A40$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0A44$_H$ - F000 0AF0$_H$	BE	BE	–
SSC0_ TSRC	SSC0 Transmit Interrupt Service Req. Control Reg.	F000 0AF4$_H$	U, SV	U, SV	0000 0000$_H$
SSC0_ RSRC	SSC0 Receive Interrupt Service Req. Control Reg.	F000 0AF8$_H$	U, SV	U, SV	0000 0000$_H$
SSC0_ ESRC	SSC0 Error Interrupt Service Req. Control Reg.	F000 0AFC$_H$	U, SV	U, SV	0000 0000$_H$
High-Speed Synchronous Serial Interface 1 (SSC1)					
SSC1_CLC	SSC1 Clock Control Register	F000 0B00$_H$	U, SV	U, SV, E	0000 0002$_H$
–	Reserved	F000 0B04$_H$	nBE	nBE	–
SSC1_ID	SSC1 Module Identification Register	F000 0B08$_H$	U, SV	BE	XXXXXXXX$_H$
–	Reserved	F000 0B0C$_H$	BE	BE	–
SSC1_CON	SSC1 Control Register	F000 0B10$_H$	U, SV	U, SV	0000 0000$_H$
SSC1_BR	SSC1 Baud Rate Timer Reload Register	F000 0B14$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0B18$_H$ - F000 0B1C$_H$	BE	BE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
SSC1_TB	SSC1 Transmit Buffer Register	F000 0B20$_H$	U, SV	U, SV	0000 0000$_H$
SSC1_RB	SSC1 Receive Buffer Register	F000 0B24$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0B28$_H$ - F000 0B2C$_H$	BE	BE	–
SSC0_ RXFCON	SSC1 Receive FIFO Control Register	F000 0B30$_H$	U, SV	U, SV	0000 0100$_H$
SSC0_ TXFCON	SSC1 Transmit FIFO Control Register	F000 0B34$_H$	U, SV	U, SV	0000 0100$_H$
SSC0_ FSTAT	SSC1 FIFO Status Register	F000 0B38$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0B3C$_H$	BE	BE	–
SSC1_ WHBCON	SSC1 Write Hardware Modified Control Register Bits	F000 0B40$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0B44$_H$ - F000 0BF0$_H$	BE	BE	–
SSC1_ TSRC	SSC1 Transmit Interrupt Service Req. Control Reg.	F000 0BF4$_H$	U, SV	U, SV	0000 0000$_H$
SSC1_ RSRC	SSC1 Receive Interrupt Service Req. Control Reg.	F000 0BF8$_H$	U, SV	U, SV	0000 0000$_H$
SSC1_ ESRC	SSC1 Error Interrupt Service Req. Control Reg.	F000 0BFC$_H$	U, SV	U, SV	0000 0000$_H$
General Purpose Timer Array (GPTA)					
GPTA_CLC	GPTA Clock Control Reg.	F000 1800$_H$	U, SV	U, SV, E	0000 0002$_H$
–	Reserved	F000 1804$_H$	nBE	nBE	–
GPTA_ID	GPTA Module Identification Register	F000 1808$_H$	U, SV	U, SV	XXXXXXXX$_H$
–	Reserved	F000 180C$_H$	nBE	nBE	–
GPTA_ SRS0	GPTA Service Request State Register 0	F000 1810$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ SRS1	GPTA Service Request State Register 1	F000 1814$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRS2	GPTA Service Request State Register 2	F000 1818$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRS3	GPTA Service Request State Register 3	F000 181C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DBGCC	GPTA Debug Clock Control Register	F000 1820$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 1824$_H$ - F000 182C$_H$	nBE	nBE	–
GPTA_ ADCCTR	GPTA AD Converter Control Register	F000 1830$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ EMGCTR0	GPTA Emergency Control Register 0	F000 1834$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ EMGCTR1	GPTA Emergency Control Register 1	F000 1838$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ MRACTL	GPTA Multiplexer Register Array Control Register	F000 183C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ MRADIN	GPTA Multiplexer Register Array Data In Register	F000 1840$_H$	U, SV, 32	U, SV, 32	0000 0000$_H$
GPTA_ MRADOUT	GPTA Multiplexer Register Array Data Out Register	F000 1844$_H$	U, SV, 32	U, SV, 32	0000 0000$_H$
–	Reserved	F000 1848$_H$ - F000 184C$_H$	nBE	nBE	–
GPTA_ FPCCTR1	GPTA Filter and Prescaler Cell Control Register 1	F000 1850$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ FPCCTR2	GPTA Filter and Prescaler Cell Control Register 2	F000 1854$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ FPCTIM0	GPTA Filter and Prescaler Cell Timer Register 0	F000 1858$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ FPCCOM0	GPTA Filter and Prescaler Cell Compare Register 0	F000 185C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ FPCTIM1	GPTA Filter and Prescaler Cell Timer Register 1	F000 1860$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ FPCCOM1	GPTA Filter and Prescaler Cell Compare Register 1	F000 1864$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ FPCTIM2	GPTA Filter and Prescaler Cell Timer Register 2	F000 1868$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ FPCCOM2	GPTA Filter and Prescaler Cell Compare Register 2	F000 186C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ FPCTIM3	GPTA Filter and Prescaler Cell Timer Register 3	F000 1870$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ FPCCOM3	GPTA Filter and Prescaler Cell Compare Register 3	F000 1874$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ FPCTIM4	GPTA Filter and Prescaler Cell Timer Register 4	F000 1878$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ FPCCOM4	GPTA Filter and Prescaler Cell Compare Register 4	F000 187C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ FPCTIM5	GPTA Filter and Prescaler Cell Timer Register 5	F000 1880$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ FPCCOM5	GPTA Filter and Prescaler Cell Compare Register 5	F000 1884$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ PDLCTR	GPTA Phase Discrimination Logic Control Register	F000 1888$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMCTR0	GPTA Duty Cycle Measurement Control Register 0	F000 188C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMTIM0	GPTA Duty Cycle Measurement Timer Register 0	F000 1890$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMCAV0	GPTA Duty Cycle Measurement Capture Register 0	F000 1894$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ DCMCOV0	GPTA Duty Cycle Measurement Capture/ Compare Register 0	F000 1898$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMCTR1	GPTA Duty Cycle Measurement Control Register 1	F000 189C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMTIM1	GPTA Duty Cycle Measurement Timer Register 1	F000 18A0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMCAV1	GPTA Duty Cycle Measurement Capture Register 1	F000 18A4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMCOV1	GPTA Duty Cycle Measurement Capture/ Compare Register 1	F000 18A8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMCTR2	GPTA Duty Cycle Measurement Control Register 2	F000 18AC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMTIM2	GPTA Duty Cycle Measurement Timer Register 2	F000 18B0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMCAV2	GPTA Duty Cycle Measurement Capture Register 2	F000 18B4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMCOV2	GPTA Duty Cycle Measurement Capture/ Compare Register 2	F000 18B8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMCTR3	GPTA Duty Cycle Measurement Control Register 3	F000 18BC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMTIM3	GPTA Duty Cycle Measurement Timer Register 3	F000 18C0$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ DCMCAV3	GPTA Duty Cycle Measurement Capture Register 3	F000 18C4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ DCMCOV3	GPTA Duty Cycle Measurement Capture/ Compare Register 3	F000 18C8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ PLLCTR	GPTA Phase Locked Loop Control Register	F000 18CC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ PLLMTI	GPTA Phase Locked Loop Microtick Register	F000 18D0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ PLLCNT	GPTA Phase Locked Loop Counter Register	F000 18D4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ PLLSTP	GPTA Phase Locked Loop Step Register	F000 18D8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ PLLREV	GPTA Phase Locked Loop Reload Register	F000 18DC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ PLLDTR	GPTA Phase Locked Loop Delta Register	F000 18E0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ CKBCTR	GPTA Clock Bus Control Register	F000 18E4$_H$	U, SV	U, SV	0000 FFFF$_H$
GPTA_ GTCTR0	GPTA Global Timer Control Register 0	F000 18E8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTREV0	GPTA Global Timer Reload Value Register 0	F000 18EC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTTIM0	GPTA Global Timer Register 0	F000 18F0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCTR1	GPTA Global Timer Control Register 1	F000 18F4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTREV1	GPTA Global Timer Reload Value Register 1	F000 18F8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTTIM1	GPTA Global Timer Register 1	F000 18FC$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ GTCCTR00	GPTA Global Timer Cell Control Register 00	F000 1900$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR00	GPTA Global Timer Cell X Register 00	F000 1904$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR01	GPTA Global Timer Cell Control Register 01	F000 1908$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR01	GPTA Global Timer Cell X Register 01	F000 190C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR02	GPTA Global Timer Cell Control Register 02	F000 1910$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR02	GPTA Global Timer Cell X Register 02	F000 1914$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR03	GPTA Global Timer Cell Control Register 03	F000 1918$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR03	GPTA Global Timer Cell X Register 03	F000 191C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR04	GPTA Global Timer Cell Control Register 04	F000 1920$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR04	GPTA Global Timer Cell X Register 04	F000 1924$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR05	GPTA Global Timer Cell Control Register 05	F000 1928$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR05	GPTA Global Timer Cell X Register 05	F000 192C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR06	GPTA Global Timer Cell Control Register 06	F000 1930$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR06	GPTA Global Timer Cell X Register 06	F000 1934$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR07	GPTA Global Timer Cell Control Register 07	F000 1938$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR07	GPTA Global Timer Cell X Register 07	F000 193C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ GTCCTR08	GPTA Global Timer Cell Control Register 08	F000 1940$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR08	GPTA Global Timer Cell X Register 08	F000 1944$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR09	GPTA Global Timer Cell Control Register 09	F000 1948$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR09	GPTA Global Timer Cell X Register 09	F000 194C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR10	GPTA Global Timer Cell Control Register 10	F000 1950$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR10	GPTA Global Timer Cell X Register 10	F000 1954$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR11	GPTA Global Timer Cell Control Register 11	F000 1958$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR11	GPTA Global Timer Cell X Register 11	F000 195C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR12	GPTA Global Timer Cell Control Register 12	F000 1960$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR12	GPTA Global Timer Cell X Register 12	F000 1964$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR13	GPTA Global Timer Cell Control Register 13	F000 1968$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR13	GPTA Global Timer Cell X Register 13	F000 196C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR14	GPTA Global Timer Cell Control Register 14	F000 1970$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR14	GPTA Global Timer Cell X Register 14	F000 1974$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR15	GPTA Global Timer Cell Control Register 15	F000 1978$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR15	GPTA Global Timer Cell X Register 15	F000 197C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ GTCCTR16	GPTA Global Timer Cell Control Register 16	F000 1980$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR16	GPTA Global Timer Cell X Register 16	F000 1984$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR17	GPTA Global Timer Cell Control Register 17	F000 1988$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR17	GPTA Global Timer Cell X Register 17	F000 198C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR18	GPTA Global Timer Cell Control Register 18	F000 1990$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR18	GPTA Global Timer Cell X Register 18	F000 1994$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR19	GPTA Global Timer Cell Control Register 19	F000 1998$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR19	GPTA Global Timer Cell X Register 19	F000 199C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR20	GPTA Global Timer Cell Control Register 20	F000 19A0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR20	GPTA Global Timer Cell X Register 20	F000 19A4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR21	GPTA Global Timer Cell Control Register 21	F000 19A8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR21	GPTA Global Timer Cell X Register 21	F000 19AC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR22	GPTA Global Timer Cell Control Register 22	F000 19B0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR22	GPTA Global Timer Cell X Register 22	F000 19B4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR23	GPTA Global Timer Cell Control Register 23	F000 19B8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR23	GPTA Global Timer Cell X Register 23	F000 19BC$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ GTCCTR24	GPTA Global Timer Cell Control Register 24	F000 19C0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR24	GPTA Global Timer Cell X Register 24	F000 19C4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR25	GPTA Global Timer Cell Control Register 25	F000 19C8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR25	GPTA Global Timer Cell X Register 25	F000 19CC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR26	GPTA Global Timer Cell Control Register 26	F000 19D0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR26	GPTA Global Timer Cell X Register 26	F000 19D4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR27	GPTA Global Timer Cell Control Register 27	F000 19D8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR27	GPTA Global Timer Cell X Register 27	F000 19DC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR28	GPTA Global Timer Cell Control Register 28	F000 19E0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR28	GPTA Global Timer Cell X Register 28	F000 19E4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR29	GPTA Global Timer Cell Control Register 29	F000 19E8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR29	GPTA Global Timer Cell X Register 29	F000 19EC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR30	GPTA Global Timer Cell Control Register 30	F000 19F0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR30	GPTA Global Timer Cell X Register 30	F000 19F4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCCTR31	GPTA Global Timer Cell Control Register 31	F000 19F8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ GTCXR31	GPTA Global Timer Cell X Register 31	F000 19FC$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ LTCCTR00	GPTA Local Timer Cell Control Register 00	F000 1A00$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR00	GPTA Local Timer Cell X Register 00	F000 1A04$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR01	GPTA Local Timer Cell Control Register 01	F000 1A08$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR01	GPTA Local Timer Cell X Register 01	F000 1A0C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR02	GPTA Local Timer Cell Control Register 02	F000 1A10$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR02	GPTA Local Timer Cell X Register 02	F000 1A14$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR03	GPTA Local Timer Cell Control Register 03	F000 1A18$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR03	GPTA Local Timer Cell X Register 03	F000 1A1C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR04	GPTA Local Timer Cell Control Register 04	F000 1A20$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR04	GPTA Local Timer Cell X Register 04	F000 1A24$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR05	GPTA Local Timer Cell Control Register 05	F000 1A28$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR05	GPTA Local Timer Cell X Register 05	F000 1A2C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR06	GPTA Local Timer Cell Control Register 06	F000 1A30$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR06	GPTA Local Timer Cell X Register 06	F000 1A34$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR07	GPTA Local Timer Cell Control Register 07	F000 1A38$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR07	GPTA Local Timer Cell X Register 07	F000 1A3C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ LTCCTR08	GPTA Local Timer Cell Control Register 08	F000 1A40$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR08	GPTA Local Timer Cell X Register 08	F000 1A44$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR09	GPTA Local Timer Cell Control Register 09	F000 1A48$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR09	GPTA Local Timer Cell X Register 09	F000 1A4C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR10	GPTA Local Timer Cell Control Register 10	F000 1A50$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR10	GPTA Local Timer Cell X Register 10	F000 1A54$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR11	GPTA Local Timer Cell Control Register 11	F000 1A58$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR11	GPTA Local Timer Cell X Register 11	F000 1A5C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR12	GPTA Local Timer Cell Control Register 12	F000 1A60$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR12	GPTA Local Timer Cell X Register 12	F000 1A64$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR13	GPTA Local Timer Cell Control Register 13	F000 1A68$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR13	GPTA Local Timer Cell X Register 13	F000 1A6C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR14	GPTA Local Timer Cell Control Register 14	F000 1A70$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR14	GPTA Local Timer Cell X Register 14	F000 1A74$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR15	GPTA Local Timer Cell Control Register 15	F000 1A78$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR15	GPTA Local Timer Cell X Register 15	F000 1A7C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ LTCCTR16	GPTA Local Timer Cell Control Register 16	F000 1A80$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR16	GPTA Local Timer Cell X Register 16	F000 1A84$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR17	GPTA Local Timer Cell Control Register 17	F000 1A88$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR17	GPTA Local Timer Cell X Register 17	F000 1A8C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR18	GPTA Local Timer Cell Control Register 18	F000 1A90$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR18	GPTA Local Timer Cell X Register 18	F000 1A94$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR19	GPTA Local Timer Cell Control Register 19	F000 1A98$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR19	GPTA Local Timer Cell X Register 19	F000 1A9C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR20	GPTA Local Timer Cell Control Register 20	F000 1AA0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR20	GPTA Local Timer Cell X Register 20	F000 1AA4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR21	GPTA Local Timer Cell Control Register 21	F000 1AA8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR21	GPTA Local Timer Cell X Register 21	F000 1AAC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR22	GPTA Local Timer Cell Control Register 22	F000 1AB0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR22	GPTA Local Timer Cell X Register 22	F000 1AB4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR23	GPTA Local Timer Cell Control Register 23	F000 1AB8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR23	GPTA Local Timer Cell X Register 23	F000 1ABC$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ LTCCTR24	GPTA Local Timer Cell Control Register 24	F000 1AC0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR24	GPTA Local Timer Cell X Register 24	F000 1AC4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR25	GPTA Local Timer Cell Control Register 25	F000 1AC8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR25	GPTA Local Timer Cell X Register 25	F000 1ACC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR26	GPTA Local Timer Cell Control Register 26	F000 1AD0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR26	GPTA Local Timer Cell X Register 26	F000 1AD4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR27	GPTA Local Timer Cell Control Register 27	F000 1AD8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR27	GPTA Local Timer Cell X Register 27	F000 1ADC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR28	GPTA Local Timer Cell Control Register 28	F000 1AE0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR28	GPTA Local Timer Cell X Register 28	F000 1AE4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR29	GPTA Local Timer Cell Control Register 29	F000 1AE8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR29	GPTA Local Timer Cell X Register 29	F000 1AEC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR30	GPTA Local Timer Cell Control Register 30	F000 1AF0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR30	GPTA Local Timer Cell X Register 30	F000 1AF4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR31	GPTA Local Timer Cell Control Register 31	F000 1AF8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR31	GPTA Local Timer Cell X Register 31	F000 1AFC$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ LTCCTR32	GPTA Local Timer Cell Control Register 32	F000 1B00$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR32	GPTA Local Timer Cell X Register 32	F000 1B04$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR33	GPTA Local Timer Cell Control Register 33	F000 1B08$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR33	GPTA Local Timer Cell X Register 33	F000 1B0C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR34	GPTA Local Timer Cell Control Register 34	F000 1B10$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR34	GPTA Local Timer Cell X Register 34	F000 1B14$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR35	GPTA Local Timer Cell Control Register 35	F000 1B18$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR35	GPTA Local Timer Cell X Register 35	F000 1B1C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR36	GPTA Local Timer Cell Control Register 36	F000 1B20$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR36	GPTA Local Timer Cell X Register 36	F000 1B24$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR37	GPTA Local Timer Cell Control Register 37	F000 1B28$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR37	GPTA Local Timer Cell X Register 37	F000 1B2C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR38	GPTA Local Timer Cell Control Register 38	F000 1B30$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR38	GPTA Local Timer Cell X Register 38	F000 1B34$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR39	GPTA Local Timer Cell Control Register 39	F000 1B38$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR39	GPTA Local Timer Cell X Register 39	F000 1B3C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ LTCCTR40	GPTA Local Timer Cell Control Register 40	F000 1B40$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR40	GPTA Local Timer Cell X Register 40	F000 1B44$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR41	GPTA Local Timer Cell Control Register 41	F000 1B48$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR41	GPTA Local Timer Cell X Register 41	F000 1B4C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR42	GPTA Local Timer Cell Control Register 42	F000 1B50$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR42	GPTA Local Timer Cell X Register 42	F000 1B54$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR43	GPTA Local Timer Cell Control Register 43	F000 1B58$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR43	GPTA Local Timer Cell X Register 43	F000 1B5C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR44	GPTA Local Timer Cell Control Register 44	F000 1B60$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR44	GPTA Local Timer Cell X Register 44	F000 1B64$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR45	GPTA Local Timer Cell Control Register 45	F000 1B68$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR45	GPTA Local Timer Cell X Register 45	F000 1B6C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR46	GPTA Local Timer Cell Control Register 46	F000 1B70$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR46	GPTA Local Timer Cell X Register 46	F000 1B74$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR47	GPTA Local Timer Cell Control Register 47	F000 1B78$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR47	GPTA Local Timer Cell X Register 47	F000 1B7C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ LTCCTR48	GPTA Local Timer Cell Control Register 48	F000 1B80$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR48	GPTA Local Timer Cell X Register 48	F000 1B84$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR49	GPTA Local Timer Cell Control Register 49	F000 1B88$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR49	GPTA Local Timer Cell X Register 49	F000 1B8C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR50	GPTA Local Timer Cell Control Register 50	F000 1B90$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR50	GPTA Local Timer Cell X Register 50	F000 1B94$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR51	GPTA Local Timer Cell Control Register 51	F000 1B98$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR51	GPTA Local Timer Cell X Register 51	F000 1B9C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR52	GPTA Local Timer Cell Control Register 52	F000 1BA0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR52	GPTA Local Timer Cell X Register 52	F000 1BA4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR53	GPTA Local Timer Cell Control Register 53	F000 1BA8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR53	GPTA Local Timer Cell X Register 53	F000 1BAC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR54	GPTA Local Timer Cell Control Register 54	F000 1BB0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR54	GPTA Local Timer Cell X Register 54	F000 1BB4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR55	GPTA Local Timer Cell Control Register 55	F000 1BB8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR55	GPTA Local Timer Cell X Register 55	F000 1BBC$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ LTCCTR56	GPTA Local Timer Cell Control Register 56	F000 1BC0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR56	GPTA Local Timer Cell X Register 56	F000 1BC4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR57	GPTA Local Timer Cell Control Register 57	F000 1BC8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR57	GPTA Local Timer Cell X Register 57	F000 1BCC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR58	GPTA Local Timer Cell Control Register 58	F000 1BD0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR58	GPTA Local Timer Cell X Register 58	F000 1BD4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR59	GPTA Local Timer Cell Control Register 59	F000 1BD8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR59	GPTA Local Timer Cell X Register 59	F000 1BDC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR60	GPTA Local Timer Cell Control Register 60	F000 1BE0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR60	GPTA Local Timer Cell X Register 60	F000 1BE4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR61	GPTA Local Timer Cell Control Register 61	F000 1BE8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR61	GPTA Local Timer Cell X Register 61	F000 1BEC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR62	GPTA Local Timer Cell Control Register 62	F000 1BF0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR62	GPTA Local Timer Cell X Register 62	F000 1BF4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCCTR63	GPTA Local Timer Cell Control Register 63	F000 1BF8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ LTCXR63	GPTA Local Timer Cell X Register 63	F000 1BFC$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
–	Reserved	$F000\ 1C00_H$ - $F000\ 1F24_H$	BE	BE	–
GPTA_ SRC53	GPTA Service Request Control Register 53	$F000\ 1F28_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC52	GPTA Service Request Control Register 52	$F000\ 1F2C_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC51	GPTA Service Request Control Register 51	$F000\ 1F30_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC50	GPTA Service Request Control Register 50	$F000\ 1F34_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC49	GPTA Service Request Control Register 49	$F000\ 1F38_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC48	GPTA Service Request Control Register 48	$F000\ 1F3C_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC47	GPTA Service Request Control Register 47	$F000\ 1F40_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC46	GPTA Service Request Control Register 46	$F000\ 1F44_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC45	GPTA Service Request Control Register 45	$F000\ 1F48_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC44	GPTA Service Request Control Register 44	$F000\ 1F4C_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC43	GPTA Service Request Control Register 43	$F000\ 1F50_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC42	GPTA Service Request Control Register 42	$F000\ 1F54_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC41	GPTA Service Request Control Register 41	$F000\ 1F58_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC40	GPTA Service Request Control Register 40	$F000\ 1F5C_H$	U, SV	U, SV	$0000\ 0000_H$
GPTA_ SRC39	GPTA Service Request Control Register 39	$F000\ 1F60_H$	U, SV	U, SV	$0000\ 0000_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ SRC38	GPTA Service Request Control Register 38	F000 1F64$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC37	GPTA Service Request Control Register 37	F000 1F68$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC36	GPTA Service Request Control Register 36	F000 1F6C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC35	GPTA Service Request Control Register 35	F000 1F70$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC34	GPTA Service Request Control Register 34	F000 1F74$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC33	GPTA Service Request Control Register 33	F000 1F78$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC32	GPTA Service Request Control Register 32	F000 1F7C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC31	GPTA Service Request Control Register 31	F000 1F80$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC30	GPTA Service Request Control Register 30	F000 1F84$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC29	GPTA Service Request Control Register 29	F000 1F88$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC28	GPTA Service Request Control Register 28	F000 1F8C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC27	GPTA Service Request Control Register 27	F000 1F90$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC26	GPTA Service Request Control Register 26	F000 1F94$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC25	GPTA Service Request Control Register 25	F000 1F98$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC24	GPTA Service Request Control Register 24	F000 1F9C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC23	GPTA Service Request Control Register 23	F000 1FA0$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ SRC22	GPTA Service Request Control Register 22	F000 1FA4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC21	GPTA Service Request Control Register 21	F000 1FA8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC20	GPTA Service Request Control Register 20	F000 1FAC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC19	GPTA Service Request Control Register 19	F000 1FB0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC18	GPTA Service Request Control Register 18	F000 1FB4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC17	GPTA Service Request Control Register 17	F000 1FB8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC16	GPTA Service Request Control Register 16	F000 1FBC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC15	GPTA Service Request Control Register 15	F000 1FC0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC14	GPTA Service Request Control Register 14	F000 1FC4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC13	GPTA Service Request Control Register 13	F000 1FC8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC12	GPTA Service Request Control Register 12	F000 1FCC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC11	GPTA Service Request Control Register 11	F000 1FD0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC10	GPTA Service Request Control Register 10	F000 1FD4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC09	GPTA Service Request Control Register 09	F000 1FD8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC08	GPTA Service Request Control Register 08	F000 1FDC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC07	GPTA Service Request Control Register 07	F000 1FE0$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
GPTA_ SRC06	GPTA Service Request Control Register 06	F000 1FE4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC05	GPTA Service Request Control Register 05	F000 1FE8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC04	GPTA Service Request Control Register 04	F000 1FEC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC03	GPTA Service Request Control Register 03	F000 1FF0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC02	GPTA Service Request Control Register 02	F000 1FF4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC01	GPTA Service Request Control Register 01	F000 1FF8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA_ SRC00	GPTA Service Request Control Register 00	F000 1FFC$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2000$_H$ - F000 21FC$_H$	BE	BE	–
Analog to Digital Converter 0 (ADC0)					
ADC0_CLC	ADC0 and ADC1 Clock Control Register	F000 2200$_H$	U, SV	U, SV, E	0000 0002$_H$
–	Reserved	F000 2204$_H$	nBE	nBE	–
ADC0_ID	ADC0 and ADC1 Module Identification Register	F000 2208$_H$	U, SV	U, SV	XXXXXXXX$_H$
–	Reserved	F000 220C$_H$	nBE	nBE	–
ADC0_ CHCON0	ADC0 Channel Control Register 0	F000 2210$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON1	ADC0 Channel Control Register 1	F000 2214$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON2	ADC0 Channel Control Register 2	F000 2218$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON3	ADC0 Channel Control Register 3	F000 221C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON4	ADC0 Channel Control Register 4	F000 2220$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ADC0_ CHCON5	ADC0 Channel Control Register 5	F000 2224$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON6	ADC0 Channel Control Register 6	F000 2228$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON7	ADC0 Channel Control Register 7	F000 222C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON8	ADC0 Channel Control Register 8	F000 2230$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON9	ADC0 Channel Control Register 9	F000 2234$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON10	ADC0 Channel Control Register 10	F000 2238$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON11	ADC0 Channel Control Register 11	F000 223C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON12	ADC0 Channel Control Register 12	F000 2240$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON13	ADC0 Channel Control Register 13	F000 2244$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON14	ADC0 Channel Control Register 14	F000 2248$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON15	ADC0 Channel Control Register 15	F000 224C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2250$_H$ - F000 2270$_H$	BE	BE	–
ADC0_ EXEV	ADC0 Source External Event Control Register	F000 2274$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ TEV	ADC0 Source Timer Event Control Register	F000 2278$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ QEV	ADC0 Source Queue Event Control Register	F000 227C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ EXEVC	ADC0 External Event Control Register	F000 2280$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ADC0_AP	ADC0 Arbitration Participation Register	F000 2284$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_SAL	ADC0 Source Arbitration Level Register	F000 2288$_H$	U, SV	U, SV	0103 4067$_H$
ADC0_TTC	ADC0 Timer Trigger Control Register	F000 228C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ EXTC0	ADC0 External Trigger Control Register 0	F000 2290$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ EXTC1	ADC0 External Trigger Control Register 1	F000 2294$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ SCON	ADC0 Source Control Register	F000 2298$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 229C$_H$	BE	BE	–
–	Reserved; these locations must not be written.	F000 22A0$_H$ - F000 22DC$_H$	nBE	nBE	–
–	Reserved	F000 22E0$_H$ - F000 22FC$_H$	BE	BE	–
ADC0_ LCCON0	ADC0 Limit Check Control Register 0	F000 2300$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ LCCON1	ADC0 Limit Check Control Register 1	F000 2304$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ LCCON2	ADC0 Limit Check Control Register 2	F000 2308$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ LCCON3	ADC0 Limit Check Control Register 3	F000 230C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2310$_H$	BE	BE	–
ADC0_ TCON	ADC0 Timer Control Reg.	F000 2314$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_CHIN	ADC0 Channel Injection Register	F000 2318$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_QR	ADC0 Queue Register	F000 231C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_CON	ADC0 Converter Control Register	F000 2320$_H$	U, SV	U, SV	0000 0001$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ADC0_SCN	ADC0 Auto Scan Control Register	F000 2324$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ REQ0	ADC0 Conversion Request Register SW0	F000 2328$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 232C$_H$	BE	BE	–
ADC0_ CHSTAT0	ADC0 Channel Status Register 0	F000 2330$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT1	ADC0 Channel Status Register 1	F000 2334$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT2	ADC0 Channel Status Register 2	F000 2338$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT3	ADC0 Channel Status Register 3	F000 233C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT4	ADC0 Channel Status Register 4	F000 2340$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT5	ADC0 Channel Status Register 5	F000 2344$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT6	ADC0 Channel Status Register 6	F000 2348$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT7	ADC0 Channel Status Register 7	F000 234C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT8	ADC0 Channel Status Register 8	F000 2350$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT9	ADC0 Channel Status Register 9	F000 2354$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT10	ADC0 Channel Status Register 10	F000 2358$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT11	ADC0 Channel Status Register 11	F000 235C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT12	ADC0 Channel Status Register 12	F000 2360$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT13	ADC0 Channel Status Register 13	F000 2364$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ADC0_ CHSTAT14	ADC0 Channel Status Register 14	F000 2368$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT15	ADC0 Channel Status Register 15	F000 236C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ QUEUE0	ADC0 Queue Status Register	F000 2370$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2374$_H$ - F000 237C$_H$	BE	BE	–
ADC0_ SW0CRP	ADC0 Software SW0 Conv. Request Pending Register	F000 2380$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2384$_H$	BE	BE	–
ADC0_ ASCRP	ADC0 Auto Scan Conversion Request Pending Register	F000 2388$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 238C$_H$	BE	BE	–
ADC0_ SYSTAT	ADC0 Synchronization Status Register	F000 2390$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2394$_H$ - F000 239C$_H$	BE	BE	–
–	Reserved; these locations must not be written.	F000 23A0$_H$ - F000 23A4$_H$	nBE	nBE	–
–	Reserved	F000 23A8$_H$ - F000 23AC$_H$	BE	BE	–
ADC0_ TSTAT	ADC0 Timer Status Register	F000 23B0$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ STAT	ADC0 Converter Status Register	F000 23B4$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ TCRP	ADC0 Timer Conversion Request Pending Register	F000 23B8$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ EXCRP	ADC0 External Conversion Request Pending Register	F000 23BC$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
–	Reserved; this location must not be written.	F000 23C0$_H$	nBE	nBE	–
–	Reserved	F000 23C4$_H$ - F000 23CC$_H$	BE	BE	–
ADC0_ MSS0	ADC0 Module Service Request Status Register 0	F000 23D0$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ MSS1	ADC0 Module Service Request Status Register 1	F000 23D4$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 23D8$_H$	BE	BE	–
ADC0_ SRNP	ADC0 Service Request Node Pointer Register	F000 23DC$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 23E0$_H$ - F000 23EC$_H$	BE	BE	–
ADC0_ SRC3	ADC0 Service Request Control Register 3	F000 23F0$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ SRC2	ADC0 Module Service Req. Control Register 2	F000 23F4$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ SRC1	ADC0 Service Request Control Register 1	F000 23F8$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ SRC0	ADC0 Service Request Control Register 0	F000 23FC$_H$	U, SV	U, SV	0000 0000$_H$
Analog to Digital Converter 1 (ADC1)					
–	Reserved	F000 2400$_H$ - F000 240C$_H$	BE	BE	–
ADC1_ CHCON0	ADC1 Channel Control Register 0	F000 2410$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON1	ADC1 Channel Control Register 1	F000 2414$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON2	ADC1 Channel Control Register 2	F000 2418$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON3	ADC1 Channel Control Register 3	F000 241C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ADC1_ CHCON4	ADC1 Channel Control Register 4	F000 2420$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON5	ADC1 Channel Control Register 5	F000 2424$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON6	ADC1 Channel Control Register 6	F000 2428$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON7	ADC1 Channel Control Register 7	F000 242C$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON8	ADC1 Channel Control Register 8	F000 2430$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON9	ADC1 Channel Control Register 9	F000 2434$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON10	ADC1 Channel Control Register 10	F000 2438$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON11	ADC1 Channel Control Register 11	F000 243C$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON12	ADC1 Channel Control Register 12	F000 2440$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON13	ADC1 Channel Control Register 13	F000 2444$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON14	ADC1 Channel Control Register 14	F000 2448$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHCON15	ADC1 Channel Control Register 15	F000 244C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2450$_H$ - F000 2470$_H$	BE	BE	–
ADC1_ EXEV	ADC0 Source External Event Control Register	F000 2474$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ TEV	ADC0 Source Timer Event Control Register	F000 2478$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ QEV	ADC0 Source Queue Event Control Register	F000 247C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ADC1_ EXEVC	ADC1 External Event Control Register	F000 2480$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_AP	ADC1 Arbitration Participation Register	F000 2484$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_SAL	ADC1 Source Arbitration Level Register	F000 2488$_H$	U, SV	U, SV	0103 4067$_H$
ADC1_TTC	ADC1 Timer Trigger Control Register	F000 248C$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ EXTC0	ADC1 External Trigger Control Register 0	F000 2490$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ EXTC1	ADC1 External Trigger Control Register 1	F000 2494$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ SCON	ADC1 Source Control Register	F000 2498$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 249C$_H$	BE	BE	–
–	Reserved; these locations must not be written.	F000 24A0$_H$ - F000 24DC$_H$	nBE	nBE	–
–	Reserved	F000 24E0$_H$ - F000 24FC$_H$	BE	BE	–
ADC1_ LCCON0	ADC1 Limit Check Control Register 0	F000 2500$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ LCCON1	ADC1 Limit Check Control Register 1	F000 2504$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ LCCON2	ADC1 Limit Check Control Register 2	F000 2508$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ LCCON3	ADC1 Limit Check Control Register 3	F000 250C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2510$_H$	BE	BE	–
ADC1_ TCON	ADC1 Timer Control Reg.	F000 2514$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_CHIN	ADC1 Channel Injection Register	F000 2518$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_QR	ADC1 Queue Register	F000 251C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ADC1_CON	ADC1 Converter Control Register	F000 2520$_H$	U, SV	U, SV	0000 0001$_H$
ADC1_SCN	ADC1 Auto Scan Control Register	F000 2524$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ REQ0	ADC1 Conversion Request Register SW0	F000 2528$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 252C$_H$	BE	BE	–
ADC1_ CHSTAT0	ADC1 Channel Status Register 0	F000 2530$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT1	ADC1 Channel Status Register 1	F000 2534$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT2	ADC1 Channel Status Register 2	F000 2538$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT3	ADC1 Channel Status Register 3	F000 253C$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT4	ADC1 Channel Status Register 4	F000 2540$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT5	ADC1 Channel Status Register 5	F000 2544$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT6	ADC1 Channel Status Register 6	F000 2548$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT7	ADC1 Channel Status Register 7	F000 254C$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT8	ADC1 Channel Status Register 8	F000 2550$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT9	ADC1 Channel Status Register 9	F000 2554$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT10	ADC1 Channel Status Register 10	F000 2558$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT11	ADC1 Channel Status Register 11	F000 255C$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT12	ADC1 Channel Status Register 12	F000 2560$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ADC1_ CHSTAT13	ADC1 Channel Status Register 13	F000 2564$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT14	ADC1 Channel Status Register 14	F000 2568$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ CHSTAT15	ADC1 Channel Status Register 15	F000 256C$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ QUEUE0	ADC1 Queue Status Register	F000 2570$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2574$_H$ - F000 257C$_H$	BE	BE	–
ADC1_ SW0CRP	ADC1 Software SW0 Conv. Request Pending Register	F000 2580$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2584$_H$	BE	BE	–
ADC1_ ASCRP	ADC1 Auto Scan Conversion Request Pending Register	F000 2588$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 258C$_H$	BE	BE	–
ADC1_ SYSTAT	ADC1 Synchronization Status Register	F000 2590$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2594$_H$ - F000 259C$_H$	BE	BE	–
–	Reserved; these locations must not be written.	F000 25A0$_H$ - F000 25A4$_H$	nBE	nBE	–
–	Reserved	F000 25A8$_H$ - F000 25AC$_H$	BE	BE	–
ADC1_ TSTAT	ADC1 Timer Status Register	F000 25B0$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ STAT	ADC1 Converter Status Register	F000 25B4$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ TCRP	ADC1 Timer Conversion Request Pending Register	F000 25B8$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ADC1_ EXCRP	ADC1 External Conversion Request Pending Register	F000 25BC$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved; this location must not be written.	F000 25C0$_H$	nBE	nBE	–
–	Reserved	F000 25C4$_H$ - F000 25CC$_H$	BE	BE	–
ADC1_ MSS0	ADC1 Module Service Request Status Register 0	F000 25D0$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ MSS1	ADC1 Module Service Request Status Register 1	F000 25D4$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 25D8$_H$	nBE	nBE	–
ADC1_ SRNP	ADC1 Service Request Node Pointer Register	F000 25DC$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 25E0$_H$ - F000 25EC$_H$	BE	BE	–
ADC1_ SRC3	ADC1 Service Request Control Register 3	F000 25F0$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ SRC2	ADC1 Service Request Control Register 2	F000 25F4$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ SRC1	ADC1 Service Request Control Register 1	F000 25F8$_H$	U, SV	U, SV	0000 0000$_H$
ADC1_ SRC0	ADC1 Service Request Control Register 0	F000 25FC$_H$	U, SV	U, SV	0000 0000$_H$
Port 0					
–	Reserved	F000 2800$_H$ - F000 280C$_H$	BE	BE	–
P0_OUT	Port 0 Data Output Reg.	F000 2810$_H$	U, SV	U, SV	0000 0000$_H$
P0_IN	Port 0 Data Input Register	F000 2814$_H$	U, SV	U, SV	0000 XXXX$_H$
P0_DIR	Port 0 Direction Register	F000 2818$_H$	U, SV	U, SV	0000 0000$_H$
P0_OD	Port 0 Open Drain Mode Register	F000 281C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
–	Reserved	F000 2820$_H$ - F000 2824$_H$	BE	BE	–
P0_ PUDSEL	Port 0 Pull-Up/Pull-Down Select Register	F000 2828$_H$	U, SV	U, SV	0000 0000$_H$
P0_ PUDEN	Port 0 Pull-Up/Pull-Down Enable Register	F000 282C$_H$	U, SV	U, SV	0000 0000$_H$
P0_ POCON	Port 0 Output Charact. Control Register	F000 2830$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2834$_H$ - F000 283C$_H$	BE	BE	–
P0_ PICON	Port 0 Input Configuration Register	F000 2840$_H$	U, SV	U, SV	0000 0000$_H$
P0_ ALTSEL0	Port 0 Alternate Select Register 0	F000 2844$_H$	U, SV	U, SV	0000 0000$_H$
P0_ ALTSEL1	Port 0 Alternate Select Register 1	F000 2848$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 284C$_H$ - F000 28FC$_H$	BE	BE	–
Port 1					
–	Reserved	F000 2900$_H$ - F000 290C$_H$	BE	BE	–
P1_OUT	Port 1 Data Output Reg.	F000 2910$_H$	U, SV	U, SV	0000 0000$_H$
P1_IN	Port 1 Data Input Register	F000 2914$_H$	U, SV	U, SV	0000 XXXX$_H$
P1_DIR	Port 1 Direction Register	F000 2918$_H$	U, SV	U, SV	0000 0000$_H$
P1_OD	Port 1 Open Drain Mode Register	F000 291C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2920$_H$ - F000 2924$_H$	BE	BE	–
P1_ PUDSEL	Port 1 Pull-Up/Pull-Down Select Register	F000 2928$_H$	U, SV	U, SV	0000 0000$_H$
P1_ PUDEN	Port 1 Pull-Up/Pull-Down Enable Register	F000 292C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
P1_ POCON	Port 1 Output Charact. Control Register	F000 2930$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2934$_H$ - F000 293C$_H$	BE	BE	–
P1_ PICON	Port 1 Input Configuration Register	F000 2940$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2944$_H$ - F000 29FC$_H$	BE	BE	–
Port 2					
–	Reserved	F000 2A00$_H$ - F000 2A0C$_H$	BE	BE	–
P2_OUT	Port 2 Data Output Reg.	F000 2A10$_H$	U, SV	U, SV	0000 0000$_H$
P2_IN	Port 2 Data Input Register	F000 2A14$_H$	U, SV	U, SV	0000 XXXX$_H$
P2_DIR	Port 2 Direction Register	F000 2A18$_H$	U, SV	U, SV	0000 0000$_H$
P2_OD	Port 2 Open Drain Mode Register	F000 2A1C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2A20$_H$ - F000 2A24$_H$	BE	BE	–
P2_ PUDSEL	Port 2 Pull-Up/Pull-Down Select Register	F000 2A28$_H$	U, SV	U, SV	0000 0000$_H$
P2_ PUDEN	Port 2 Pull-Up/Pull-Down Enable Register	F000 2A2C$_H$	U, SV	U, SV	0000 0000$_H$
P2_ POCON	Port 2 Output Charact. Control Register	F000 2A30$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2A34$_H$ - F000 2A3C$_H$	BE	BE	–
P2_ PICON	Port 2 Input Configuration Register	F000 2A40$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2A44$_H$ - F000 2AFC$_H$	BE	BE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
Port 3					
–	Reserved	F000 2B00$_H$ - F000 2B0C$_H$	BE	BE	–
P3_OUT	Port 3 Data Output Reg.	F000 2B10$_H$	U, SV	U, SV	0000 0000$_H$
P3_IN	Port 3 Data Input Register	F000 2B14$_H$	U, SV	U, SV	0000 XXXX$_H$
P3_DIR	Port 3 Direction Register	F000 2B18$_H$	U, SV	U, SV	0000 0000$_H$
P3_OD	Port 3 Open Drain Mode Register	F000 2B1C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2B20$_H$ - F000 2B24$_H$	BE	BE	–
P3_ PUDSEL	Port 3 Pull-Up/Pull-Down Select Register	F000 2B28$_H$	U, SV	U, SV	0000 0000$_H$
P3_ PUDEN	Port 3 Pull-Up/Pull-Down Enable Register	F000 2B2C$_H$	U, SV	U, SV	0000 0000$_H$
P3_ POCON	Port 3 Output Charact. Control Register	F000 2B30$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2B34$_H$ - F000 2B3C$_H$	BE	BE	–
P3_ PICON	Port 3 Input Configuration Register	F000 2B40$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2B44$_H$ - F000 2BFC$_H$	BE	BE	–
Port 4					
–	Reserved	F000 2C00$_H$ - F000 2C0C$_H$	BE	BE	–
P4_OUT	Port 4 Data Output Reg.	F000 2C10$_H$	U, SV	U, SV	0000 0000$_H$
P4_IN	Port 4 Data Input Register	F000 2C14$_H$	U, SV	U, SV	0000 XXXX$_H$
P4_DIR	Port 4 Direction Register	F000 2C18$_H$	U, SV	U, SV	0000 0000$_H$
P4_OD	Port 4 Open Drain Mode Register	F000 2C1C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2C20$_H$ - F000 2C24$_H$	BE	BE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
P4_ PUDSEL	Port 4 Pull-Up/Pull-Down Select Register	F000 2C28$_H$	U, SV	U, SV	0000 0000$_H$
P4_ PUDEN	Port 4 Pull-Up/Pull-Down Enable Register	F000 2C2C$_H$	U, SV	U, SV	0000 0000$_H$
P4_ POCON	Port 4 Output Charact. Control Register	F000 2C30$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2C34$_H$ - F000 2C3C$_H$	BE	BE	–
P4_ PICON	Port 4 Input Configuration Register	F000 2C40	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2C44$_H$ - F000 2CFC$_H$	BE	BE	–
Port 5					
–	Reserved	F000 2D00$_H$ - F000 2D0C$_H$	BE	BE	–
P5_OUT	Port 5 Data Output Reg.	F000 2D10$_H$	U, SV	U, SV	0000 0000$_H$
P5_IN	Port 5 Data Input Register	F000 2D14$_H$	U, SV	U, SV	0000 XXXX$_H$
P5_DIR	Port 5 Direction Register	F000 2D18$_H$	U, SV	U, SV	0000 0000$_H$
P5_OD	Port 5 Open Drain Mode Register	F000 2D1C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2D20$_H$ - F000 2D24$_H$	BE	BE	–
P5_ PUDSEL	Port 5 Pull-Up/Pull-Down Select Register	F000 2D28$_H$	U, SV	U, SV	0000 0000$_H$
P5_ PUDEN	Port 5 Pull-Up/Pull-Down Enable Register	F000 2D2C$_H$	U, SV	U, SV	0000 0000$_H$
P5_ POCON	Port 5 Output Charact. Control Register	F000 2D30$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2D34$_H$ - F000 2D3C$_H$	BE	BE	–
P5_ PICON	Port 5 Input Configuration Register	F000 2D40$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
P5_ ALTSEL0	Port 5 Alternate Select Register 0	F000 2D44$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 2D48$_H$ - F000 2DFC$_H$	BE	BE	–
DMA Controller					
DMA_CLC	DMA Clock Control Reg.	F000 3C00$_H$	U, SV	U, SV, E	0000 0003$_H$
–	Reserved	F000 3C04$_H$	BE	BE	–
DMA_ID	DMA Controller Identification Register	F000 3C08$_H$	U, SV	BE	XXXXXXXX$_H$
–	Reserved	F000 3C0C$_H$	BE	BE	–
DMA_ GCTR	DMA Global Control and Status Register	F000 3C10$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 3C14$_H$ - F000 3C1C$_H$	BE	BE	–
DMA_ CSR00	DMA Channel 00 Control and Status Register	F000 3C20$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ OTC00	DMA Channel 00 Offset and Transfer Count Reg.	F000 3C24$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SSA00	DMA Channel 00 Source Start Address Register	F000 3C28$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SEA00	DMA Channel 00 Source End Address Register	F000 3C2C$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DSA00	DMA Channel 00 Dest. Start Address Register	F000 3C30$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DEA00	DMA Channel 00 Dest. End Address Register	F000 3C34$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 3C38$_H$ - F000 3C3C$_H$	BE	BE	–
DMA_ CSR01	DMA Channel 01 Control and Status Register	F000 3C40$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ OTC01	DMA Channel 01 Offset and Transfer Count Reg.	F000 3C44$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
DMA_ SSA01	DMA Channel 01 Source Start Address Register	F000 3C48$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SEA01	DMA Channel 01 Source End Address Register	F000 3C4C$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DSA01	DMA Channel 01 Dest. Start Address Register	F000 3C50$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DEA01	DMA Channel 01 Dest. End Address Register	F000 3C54$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 3C58$_H$ - F000 3C5C$_H$	BE	BE	–
DMA_ CSR02	DMA Channel 02 Control and Status Register	F000 3C60$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ OTC02	DMA Channel 02 Offset and Transfer Count Reg.	F000 3C64$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SSA02	DMA Channel 02 Source Start Address Register	F000 3C68$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SEA02	DMA Channel 02 Source End Address Register	F000 3C6C$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DSA02	DMA Channel 02 Dest. Start Address Register	F000 3C70$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DEA02	DMA Channel 02 Dest. End Address Register	F000 3C74$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 3C78$_H$ - F000 3C7C$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ CSR03	DMA Channel 03 Control and Status Register	F000 3C80$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ OTC03	DMA Channel 03 Offset and Transfer Count Reg.	F000 3C84$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SSA03	DMA Channel 03 Source Start Address Register	F000 3C88$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SEA03	DMA Channel 03 Source End Address Register	F000 3C8C$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
DMA_ DSA03	DMA Channel 03 Dest. Start Address Register	F000 3C90$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DEA03	DMA Channel 03 Dest. End Address Register	F000 3C94$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 3C98$_H$ - F000 3C9C$_H$	BE	BE	–
DMA_ CSR10	DMA Channel 10 Control and Status Register	F000 3CA0$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ OTC10	DMA Channel 10 Offset and Transfer Count Reg.	F000 3CA4$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SSA10	DMA Channel 10 Source Start Address Register	F000 3CA8$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SEA10	DMA Channel 10 Source End Address Register	F000 3CAC$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DSA10	DMA Channel 10 Dest. Start Address Register	F000 3CB0$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DEA10	DMA Channel 10 Dest. End Address Register	F000 3CB4$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 3CB8$_H$ - F000 3CBC$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ CSR11	DMA Channel 11 Control and Status Register	F000 3CC0$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ OTC11	DMA Channel 11 Offset and Transfer Count Reg.	F000 3CC4$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SSA11	DMA Channel 11 Source Start Address Register	F000 3CC8$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SEA11	DMA Channel 11 Source End Address Register	F000 3CCC$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DSA11	DMA Channel 11 Dest. Start Address Register	F000 3CD0$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DEA11	DMA Channel 11 Dest. End Address Register	F000 3CD4$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
–	Reserved	F000 3CD8$_H$ - F000 3CDC$_H$	BE	BE	–
DMA_ CSR12	DMA Channel 12 Control and Status Register	F000 3CE0$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ OTC12	DMA Channel 12 Offset and Transfer Count Reg.	F000 3CE4$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SSA12	DMA Channel 12 Source Start Address Register	F000 3CE8$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SEA12	DMA Channel 12 Source End Address Register	F000 3CEC$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DSA12	DMA Channel 12 Dest. Start Address Register	F000 3CF0$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DEA12	DMA Channel 12 Dest. End Address Register	F000 3CF4$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 3CF8$_H$ - F000 3CFC$_H$	BE	BE	–
DMA_ CSR13	DMA Channel 13 Control and Status Register	F000 3D00$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ OTC13	DMA Channel 13 Offset and Transfer Count Reg.	F000 3D04$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SSA13	DMA Channel 13 Source Start Address Register	F000 3D08$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ SEA13	DMA Channel 13 Source End Address Register	F000 3D0C$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DSA13	DMA Channel 13 Dest. Start Address Register	F000 3D10$_H$	U, SV	U, SV	0000 0000$_H$
DMA_ DEA13	DMA Channel 13 Dest. End Address Register	F000 3D14$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 3D18$_H$ - F000 3DDC$_H$	BE	BE	–
DMA_SRC7	DMA Service Request Control Register 7	F000 3DE0$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
DMA_SRC6	DMA Service Request Control Register 6	F000 3DE4$_H$	U, SV	U, SV	0000 0000$_H$
DMA_SRC5	DMA Service Request Control Register 5	F000 3DE8$_H$	U, SV	U, SV	0000 0000$_H$
DMA_SRC4	DMA Service Request Control Register 4	F000 3DEC$_H$	U, SV	U, SV	0000 0000$_H$
DMA_SRC3	DMA Service Request Control Register 3	F000 3DF0$_H$	U, SV	U, SV	0000 0000$_H$
DMA_SRC2	DMA Service Request Control Register 2	F000 3DF4$_H$	U, SV	U, SV	0000 0000$_H$
DMA_SRC1	DMA Service Request Control Register 1	F000 3DF8$_H$	U, SV	U, SV	0000 0000$_H$
DMA_SRC0	DMA Service Request Control Register 0	F000 3DFC$_H$	U, SV	U, SV	0000 0000$_H$
Controller Area Network Module (CAN)					
CAN_CLC	CAN Clock Control Reg.	F010 0000$_H$	U, SV	U, SV, E	0000 0002$_H$
–	Reserved; this location must not be written.	F010 0004$_H$	nBE	nBE	–
CAN_ID	CAN Module Identification Register	F010 0008$_H$	U, SV	U, SV	XXXXXXXX$_H$
–	Reserved	F010 000C$_H$ - F010 01FC$_H$	nBE	nBE	–
CAN_ACR	CAN Node A Control Reg.	F010 0200$_H$	U, SV	U, SV	0000 0001$_H$
CAN_ASR	CAN Node A Status Reg.	F010 0204$_H$	U, SV	U, SV	0000 0000$_H$
CAN_AIR	CAN Node A Interrupt Pending Register	F010 0208$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ABTR	CAN Node A Bit Timing Register	F010 020C$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ AGINP	CAN Node A Global Int. Node Pointer Register	F010 0210$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ AFCR	CAN Node A Frame Counter Register	F010 0214$_H$	U, SV	U, SV	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ AIMR0	CAN Node A INTID Mask Register 0	F010 0218$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ AIMR4	CAN Node A INTID Mask Register 4	F010 021C$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ AECNT	CAN Node A Error Counter Register	F010 0220$_H$	U, SV	U, SV	0060 0000$_H$
–	Reserved	F010 0224$_H$ - F010 023C$_H$	nBE	nBE	–
CAN_BCR	CAN Node B Control Reg.	F010 0240$_H$	U, SV	U, SV	0000 0001$_H$
CAN_BSR	CAN Node B Status Reg.	F010 0244$_H$	U, SV	U, SV	0000 0000$_H$
CAN_BIR	CAN Node B Interrupt Pending Register	F010 0248$_H$	U, SV	U, SV	0000 0000$_H$
CAN_BBTR	CAN Node B Bit Timing Register	F010 024C$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ BGINP	CAN Node B Global Int. Node Pointer Register	F010 0250$_H$	U, SV	U, SV	0000 0000$_H$
CAN_BFCR	CAN Node B Frame Counter Register	F010 0254$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ BIMR0	CAN Node B INTID Mask Register 0	F010 0258$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ BIMR4	CAN Node B INTID Mask Register 4	F010 025C$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ BECNT	CAN Node B Error Counter Register	F010 0260$_H$	U, SV	U, SV	0060 0000$_H$
–	Reserved	F010 0264$_H$ - F010 0280$_H$	nBE	nBE	–
CAN_ RXIPND	CAN Receive Interrupt Pending Register	F010 0284$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ TXIPND	CAN Transmit Interrupt Pending Register	F010 0288$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 028C$_H$ - F010 02FC$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR00	CAN Message Object 0 Data Register 0	F010 0300$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR04	CAN Message Object 0 Data Register 4	F010 0304$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR0	CAN Message Object 0 Arbitration Register	F010 0308$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR0	CAN Message Object 0 Acceptance Mask Reg.	F010 030C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR0	CAN Message Object 0 Message Control Register	F010 0310$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG0	CAN Message Object 0 Message Config. Register	F010 0314$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGFGCR0	CAN Message Object 0 FIFO/Gateway Control Register	F010 0318$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 031C$_H$	nBE	nBE	–
CAN_ MSGDR10	CAN Message Object 1 Data Register 0	F010 0320$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR14	CAN Message Object 1 Data Register 4	F010 0324$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR1	CAN Message Object 1 Arbitration Register	F010 0328$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR1	CAN Message Object 1 Acceptance Mask Reg.	F010 032C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR1	CAN Message Object 1 Message Control Register	F010 0330$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG1	CAN Message Object 1 Message Config. Register	F010 0334$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGFGCR1	CAN Message Object 1 FIFO/Gateway Control Register	F010 0338$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 033C$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR20	CAN Message Object 2 Data Register 0	F010 0340$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR24	CAN Message Object 2 Data Register 4	F010 0344$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR2	CAN Message Object 2 Arbitration Register	F010 0348$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR2	CAN Message Object 2 Acceptance Mask Reg.	F010 034C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR2	CAN Message Object 2 Message Control Register	F010 0350$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG2	CAN Message Object 2 Message Config. Register	F010 0354$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGFGCR2	CAN Message Object 2 FIFO/Gateway Control Register	F010 0358$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 035C$_H$	nBE	nBE	–
CAN_ MSGDR30	CAN Message Object 3 Data Register 0	F010 0360$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR34	CAN Message Object 3 Data Register 4	F010 0364$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR3	CAN Message Object 3 Arbitration Register	F010 0368$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR3	CAN Message Object 3 Acceptance Mask Reg.	F010 036C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR3	CAN Message Object 3 Message Control Register	F010 0370$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG3	CAN Message Object 3 Message Config. Register	F010 0374$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGFGCR3	CAN Message Object 3 FIFO/Gateway Control Register	F010 0378$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 037C$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR40	CAN Message Object 4 Data Register 0	F010 0380$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR44	CAN Message Object 4 Data Register 4	F010 0384$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR4	CAN Message Object 4 Arbitration Register	F010 0388$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR4	CAN Message Object 4 Acceptance Mask Reg.	F010 038C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR4	CAN Message Object 4 Message Control Register	F010 0390$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG4	CAN Message Object 4 Message Config. Register	F010 0394$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGFGCR4	CAN Message Object 4 FIFO/Gateway Control Register	F010 0398$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 039C$_H$	nBE	nBE	–
CAN_ MSGDR50	CAN Message Object 5 Data Register 0	F010 03A0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR54	CAN Message Object 5 Data Register 4	F010 03A4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR5	CAN Message Object 5 Arbitration Register	F010 03A8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR5	CAN Message Object 5 Acceptance Mask Reg.	F010 03AC$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR5	CAN Message Object 5 Message Control Register	F010 03B0$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG5	CAN Message Object 5 Message Config. Register	F010 03B4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGFGCR5	CAN Message Object 5 FIFO/Gateway Control Register	F010 03B8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 03BC$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR60	CAN Message Object 6 Data Register 0	F010 03C0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR64	CAN Message Object 6 Data Register 4	F010 03C4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR6	CAN Message Object 6 Arbitration Register	F010 03C8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR6	CAN Message Object 6 Acceptance Mask Reg.	F010 03CC$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR6	CAN Message Object 6 Message Control Register	F010 03D0$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG6	CAN Message Object 6 Message Config. Register	F010 03D4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGFGCR6	CAN Message Object 6 FIFO/Gateway Control Register	F010 03D8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 03DC$_H$	nBE	nBE	–
CAN_ MSGDR70	CAN Message Object 7 Data Register 0	F010 03E0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR74	CAN Message Object 7 Data Register 4	F010 03E4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR7	CAN Message Object 7 Arbitration Register	F010 03E8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR7	CAN Message Object 7 Acceptance Mask Reg.	F010 03EC$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR7	CAN Message Object 7 Message Control Register	F010 03F0$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG7	CAN Message Object 7 Message Config. Register	F010 03F4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGFGCR7	CAN Message Object 7 FIFO/Gateway Control Register	F010 03F8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 03FC$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR80	CAN Message Object 8 Data Register 0	F010 0400$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR84	CAN Message Object 8 Data Register 4	F010 0404$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR8	CAN Message Object 8 Arbitration Register	F010 0408$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR8	CAN Message Object 8 Acceptance Mask Reg.	F010 040C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR8	CAN Message Object 8 Message Control Register	F010 0410$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG8	CAN Message Object 8 Message Config. Register	F010 0414$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGFGCR8	CAN Message Object 8 FIFO/Gateway Control Register	F010 0418$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 041C$_H$	nBE	nBE	–
CAN_ MSGDR90	CAN Message Object 9 Data Register 0	F010 0420$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR94	CAN Message Object 9 Data Register 4	F010 0424$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR9	CAN Message Object 9 Arbitration Register	F010 0428$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR9	CAN Message Object 9 Acceptance Mask Reg.	F010 042C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR9	CAN Message Object 9 Message Control Register	F010 0430$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG9	CAN Message Object 9 Message Config. Register	F010 0434$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGFGCR9	CAN Message Object 9 FIFO/Gateway Control Register	F010 0438$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 043C$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR100	CAN Message Object 10 Data Register 0	F010 0440$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR104	CAN Message Object 10 Data Register 4	F010 0444$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR10	CAN Message Object 10 Arbitration Register	F010 0448$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR10	CAN Message Object 10 Acceptance Mask Reg.	F010 044C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR10	CAN Message Object 10 Message Control Register	F010 0450$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG10	CAN Message Object 10 Message Config. Register	F010 0454$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR10	CAN Message Object 10 FIFO/Gateway Control Register	F010 0458$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 045C$_H$	nBE	nBE	–
CAN_ MSGDR110	CAN Message Object 11 Data Register 0	F010 0460$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR114	CAN Message Object 11 Data Register 4	F010 0464$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR11	CAN Message Object 11 Arbitration Register	F010 0468$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR11	CAN Message Object 11 Acceptance Mask Reg.	F010 046C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR11	CAN Message Object 11 Message Control Register	F010 0470$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG11	CAN Message Object 11 Message Config. Register	F010 0474$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR11	CAN Message Object 11 FIFO/Gateway Control Register	F010 0478$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 047C$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR120	CAN Message Object 12 Data Register 0	F010 0480$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR124	CAN Message Object 12 Data Register 4	F010 0484$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR12	CAN Message Object 12 Arbitration Register	F010 0488$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR12	CAN Message Object 12 Acceptance Mask Reg.	F010 048C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR12	CAN Message Object 12 Message Control Register	F010 0490$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG12	CAN Message Object 12 Message Config. Register	F010 0494$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR12	CAN Message Object 12 FIFO/Gateway Control Register	F010 0498$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 049C$_H$	nBE	nBE	–
CAN_ MSGDR130	CAN Message Object 13 Data Register 0	F010 04A0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR134	CAN Message Object 13 Data Register 4	F010 04A4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR13	CAN Message Object 13 Arbitration Register	F010 04A8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR13	CAN Message Object 13 Acceptance Mask Reg.	F010 04AC$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR13	CAN Message Object 13 Message Control Register	F010 04B0$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG13	CAN Message Object 13 Message Config. Register	F010 04B4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR13	CAN Message Object 13 FIFO/Gateway Control Register	F010 04B8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 04BC$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR140	CAN Message Object 14 Data Register 0	F010 04C0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR144	CAN Message Object 14 Data Register 4	F010 04C4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR14	CAN Message Object 14 Arbitration Register	F010 04C8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR14	CAN Message Object 14 Acceptance Mask Reg.	F010 04CC$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR14	CAN Message Object 14 Message Control Register	F010 04D0$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG14	CAN Message Object 14 Message Config. Register	F010 04D4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR14	CAN Message Object 14 FIFO/Gateway Control Register	F010 04D8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 04DC$_H$	nBE	nBE	–
CAN_ MSGDR150	CAN Message Object 15 Data Register 0	F010 04E0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR154	CAN Message Object 15 Data Register 4	F010 04E4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR15	CAN Message Object 15 Arbitration Register	F010 04E8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR15	CAN Message Object 15 Acceptance Mask Reg.	F010 04EC$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR15	CAN Message Object 15 Message Control Register	F010 04F0$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG15	CAN Message Object 15 Message Config. Register	F010 04F4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR15	CAN Message Object 15 FIFO/Gateway Control Register	F010 04F8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 04FC$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR160	CAN Message Object 16 Data Register 0	F010 0500$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR164	CAN Message Object 16 Data Register 4	F010 0504$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR16	CAN Message Object 16 Arbitration Register	F010 0508$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR16	CAN Message Object 16 Acceptance Mask Reg.	F010 050C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR16	CAN Message Object 16 Message Control Register	F010 0510$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG16	CAN Message Object 16 Message Config. Register	F010 0514$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR16	CAN Message Object 16 FIFO/Gateway Control Register	F010 0518$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 051C$_H$	nBE	nBE	–
CAN_ MSGDR170	CAN Message Object 17 Data Register 0	F010 0520$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR174	CAN Message Object 17 Data Register 4	F010 0524$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR17	CAN Message Object 17 Arbitration Register	F010 0528$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR17	CAN Message Object 17 Acceptance Mask Reg.	F010 052C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR17	CAN Message Object 17 Message Control Register	F010 0530$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG17	CAN Message Object 17 Message Config. Register	F010 0534$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR17	CAN Message Object 17 FIFO/Gateway Control Register	F010 0538$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 053C$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR180	CAN Message Object 18 Data Register 0	F010 0540$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR184	CAN Message Object 18 Data Register 4	F010 0544$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR18	CAN Message Object 18 Arbitration Register	F010 0548$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR18	CAN Message Object 18 Acceptance Mask Reg.	F010 054C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR18	CAN Message Object 18 Message Control Register	F010 0550$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG18	CAN Message Object 18 Message Config. Register	F010 0554$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR18	CAN Message Object 18 FIFO/Gateway Control Register	F010 0558$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 055C$_H$	nBE	nBE	–
CAN_ MSGDR190	CAN Message Object 19 Data Register 0	F010 0560$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR194	CAN Message Object 19 Data Register 4	F010 0564$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR19	CAN Message Object 19 Arbitration Register	F010 0568$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR19	CAN Message Object 19 Acceptance Mask Reg.	F010 056C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR19	CAN Message Object 19 Message Control Register	F010 0570$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG19	CAN Message Object 19 Message Config. Register	F010 0574$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR19	CAN Message Object 19 FIFO/Gateway Control Register	F010 0578$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 057C$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR200	CAN Message Object 20 Data Register 0	F010 0580$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR204	CAN Message Object 20 Data Register 4	F010 0584$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR20	CAN Message Object 20 Arbitration Register	F010 0588$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR20	CAN Message Object 20 Acceptance Mask Reg.	F010 058C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR20	CAN Message Object 20 Message Control Register	F010 0590$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG20	CAN Message Object 20 Message Config. Register	F010 0594$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR20	CAN Message Object 20 FIFO/Gateway Control Register	F010 0598$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 059C$_H$	nBE	nBE	–
CAN_ MSGDR210	CAN Message Object 21 Data Register 0	F010 05A0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR214	CAN Message Object 21 Data Register 4	F010 05A4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR21	CAN Message Object 21 Arbitration Register	F010 05A8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR21	CAN Message Object 21 Acceptance Mask Reg.	F010 05AC$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR21	CAN Message Object 21 Message Control Register	F010 05B0$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG21	CAN Message Object 21 Message Config. Register	F010 05B4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR21	CAN Message Object 21 FIFO/Gateway Control Register	F010 05B8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 05BC$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR220	CAN Message Object 22 Data Register 0	F010 05C0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR224	CAN Message Object 22 Data Register 4	F010 05C4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR22	CAN Message Object 22 Arbitration Register	F010 05C8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR22	CAN Message Object 22 Acceptance Mask Reg.	F010 05CC$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR22	CAN Message Object 22 Message Control Register	F010 05D0$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG22	CAN Message Object 22 Message Config. Register	F010 05D4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR22	CAN Message Object 22 FIFO/Gateway Control Register	F010 05D8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 05DC$_H$	nBE	nBE	–
CAN_ MSGDR230	CAN Message Object 23 Data Register 0	F010 05E0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR234	CAN Message Object 23 Data Register 4	F010 05E4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR23	CAN Message Object 23 Arbitration Register	F010 05E8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR23	CAN Message Object 23 Acceptance Mask Reg.	F010 05EC$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR23	CAN Message Object 23 Message Control Register	F010 05F0$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG23	CAN Message Object 23 Message Config. Register	F010 05F4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR23	CAN Message Object 23 FIFO/Gateway Control Register	F010 05F8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 05FC$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR240	CAN Message Object 24 Data Register 0	F010 0600$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR244	CAN Message Object 24 Data Register 4	F010 0604$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR24	CAN Message Object 24 Arbitration Register	F010 0608$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR24	CAN Message Object 24 Acceptance Mask Reg.	F010 060C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR24	CAN Message Object 24 Message Control Register	F010 0610$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG24	CAN Message Object 24 Message Config. Register	F010 0614$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR24	CAN Message Object 24 FIFO/Gateway Control Register	F010 0618$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 061C$_H$	nBE	nBE	–
CAN_ MSGDR250	CAN Message Object 25 Data Register 0	F010 0620$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR254	CAN Message Object 25 Data Register 4	F010 0624$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR25	CAN Message Object 25 Arbitration Register	F010 0628$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR25	CAN Message Object 25 Acceptance Mask Reg.	F010 062C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR25	CAN Message Object 25 Message Control Register	F010 0630$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG25	CAN Message Object 25 Message Config. Register	F010 0634$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR25	CAN Message Object 25 FIFO/Gateway Control Register	F010 0638$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 063C$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR260	CAN Message Object 26 Data Register 0	F010 0640$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR264	CAN Message Object 26 Data Register 4	F010 0644$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR26	CAN Message Object 26 Arbitration Register	F010 0648$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR26	CAN Message Object 26 Acceptance Mask Reg.	F010 064C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR26	CAN Message Object 26 Message Control Register	F010 0650$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG26	CAN Message Object 26 Message Config. Register	F010 0654$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR26	CAN Message Object 26 FIFO/Gateway Control Register	F010 0658$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 065C$_H$	nBE	nBE	–
CAN_ MSGDR270	CAN Message Object 27 Data Register 0	F010 0660$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR274	CAN Message Object 27 Data Register 4	F010 0664$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR27	CAN Message Object 27 Arbitration Register	F010 0668$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR27	CAN Message Object 27 Acceptance Mask Reg.	F010 066C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR27	CAN Message Object 27 Message Control Register	F010 0670$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG27	CAN Message Object 27 Message Config. Register	F010 0674$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR27	CAN Message Object 27 FIFO/Gateway Control Register	F010 0678$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 067C$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR280	CAN Message Object 28 Data Register 0	F010 0680$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR284	CAN Message Object 28 Data Register 4	F010 0684$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR28	CAN Message Object 28 Arbitration Register	F010 0688$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR28	CAN Message Object 28 Acceptance Mask Reg.	F010 068C$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR28	CAN Message Object 28 Message Control Register	F010 0690$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG28	CAN Message Object 28 Message Config. Register	F010 0694$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR28	CAN Message Object 28 FIFO/Gateway Control Register	F010 0698$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 069C$_H$	nBE	nBE	–
CAN_ MSGDR290	CAN Message Object 29 Data Register 0	F010 06A0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR294	CAN Message Object 29 Data Register 4	F010 06A4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR29	CAN Message Object 29 Arbitration Register	F010 06A8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGMR29	CAN Message Object 29 Acceptance Mask Reg.	F010 06AC$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR29	CAN Message Object 29 Message Control Register	F010 06B0$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG29	CAN Message Object 29 Message Config. Register	F010 06B4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR29	CAN Message Object 29 FIFO/Gateway Control Register	F010 06B8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 06BC$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_ MSGDR300	CAN Message Object 30 Data Register 0	F010 06C0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR304	CAN Message Object 30 Data Register 4	F010 06C4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR30	CAN Message Object 30 Arbitration Register	F010 06C8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR30	CAN Message Object 30 Acceptance Mask Reg.	F010 06CC$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR30	CAN Message Object 30 Message Control Register	F010 06D0$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG30	CAN Message Object 30 Message Config. Register	F010 06D4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR30	CAN Message Object 30 FIFO/Gateway Control Register	F010 06D8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 06DC$_H$	nBE	nBE	–
CAN_ MSGDR310	CAN Message Object 31 Data Register 0	F010 06E0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGDR314	CAN Message Object 31 Data Register 4	F010 06E4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAR31	CAN Message Object 31 Arbitration Register	F010 06E8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSGAMR31	CAN Message Object 31 Acceptance Mask Reg.	F010 06EC$_H$	U, SV	U, SV	FFFF FFFF$_H$
CAN_ MSGCTR31	CAN Message Object 31 Message Control Register	F010 06F0$_H$	U, SV	U, SV	0000 5555$_H$
CAN_ MSGCFG31	CAN Message Object 31 Message Config. Register	F010 06F4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_MSG FGCR31	CAN Message Object 31 FIFO/Gateway Control Register	F010 06F8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 06FC$_H$ - F010 0ADC$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CAN_SRC7	CAN Service Request Control Register 7	F010 0AE0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_SRC6	CAN Service Request Control Register 6	F010 0AE4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_SRC5	CAN Service Request Control Register 5	F010 0AE8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_SRC4	CAN Service Request Control Register 4	F010 0AEC$_H$	U, SV	U, SV	0000 0000$_H$
CAN_SRC3	CAN Service Request Control Register 3	F010 0AF0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_SRC2	CAN Service Request Control Register 2	F010 0AF4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_SRC1	CAN Service Request Control Register 1	F010 0AF8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_SRC0	CAN Service Request Control Register 0	F010 0AFC$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 0B00$_H$ - F010 0BFF$_H$	nBE	nBE	–
CPU Slave Interface Registers (CPS)					
–	Reserved	FFFE FF00$_H$ - FFFE FF04$_H$	BE	BE	–
CPU_ID	CPU Module Identification Register	FFFE FF08$_H$	U, SV	BE	0000 0202$_H$
–	Reserved	FFFE FF0C$_H$ - FFFE FFB8$_H$	BE	BE	–
SBSRC0	Software Break Service Request Control Reg. 0	FFFE FFBC$_H$	U, SV	SV	0000 0000$_H$
–	Reserved	FFFE FFC0$_H$ - FFFE FFEC$_H$	BE	BE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CPU_SRC3	CPU Service Request Control Register 3	FFFE FFF0$_H$	U, SV	SV	0000 0000$_H$
CPU_SRC2	CPU Service Request Control Register 2	FFFE FFF4$_H$	U, SV	SV	0000 0000$_H$
CPU_SRC1	CPU Service Request Control Register 1	FFFE FFF8$_H$	U, SV	SV	0000 0000$_H$
CPU_SRC0	CPU Service Request Control Register 0	FFFE FFFC$_H$	U, SV	SV	0000 0000$_H$
Memory Protection Registers					
DPR0_0L	Data Seg. Protect. Reg. Set 0, Range 0, Lower	FFFF C000$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_0U	Data Seg. Protect. Reg. Set 0, Range 0, Upper	FFFF C004$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_1L	Data Seg. Protect. Reg. Set 0, Range 1, Lower	FFFF C008$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_1U	Data Seg. Protect. Reg. Set 0, Range 1, Upper	FFFF C00C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_2L	Data Seg. Protect. Reg. Set 0, Range 2, Lower	FFFF C010$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_2U	Data Seg. Protect. Reg. Set 0, Range 2, Upper	FFFF C014$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_3L	Data Seg. Protect. Reg. Set 0, Range 3, Lower	FFFF C018$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_3U	Data Seg. Protect. Reg. Set 0, Range 3, Upper	FFFF C01C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	FFFF C020$_H$ - FFFF C3FC$_H$	nE	nE	–
DPR1_0L	Data Seg. Protect. Reg. Set 1, Range 0, Lower	FFFF C400$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR1_0U	Data Seg. Protect. Reg. Set 1, Range 0, Upper	FFFF C404$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR1_1L	Data Seg. Protect. Reg. Set 1, Range 1, Lower	FFFF C408$_H$	U, SV, 32	SV, 32	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
DPR1_1U	Data Seg. Protect. Reg. Set 1, Range 1, Upper	FFFF C40C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR1_2L	Data Seg. Protect. Reg. Set 1, Range 2, Lower	FFFF C410$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR1_2U	Data Seg. Protect. Reg. Set 1, Range 2, Upper	FFFF C414$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR1_3L	Data Seg. Protect. Reg. Set 1, Range 3, Lower	FFFF C418$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR1_3U	Data Seg. Protect. Reg. Set 1, Range 3, Upper	FFFF C41C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	FFFF C420$_H$ - FFFF CFFC$_H$	nE	nE	–
CPR0_0L	Code Seg. Prot. Register Set 0, Range 0, Lower	FFFF D000$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPR0_0U	Code Seg. Prot. Register Set 0, Range 0, Upper	FFFF D004$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPR0_1L	Code Seg. Prot. Register Set 0, Range 1, Lower	FFFF D008$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPR0_1U	Code Seg. Prot. Register Set 0, Range 1, Upper	FFFF D00C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	FFFF D010$_H$ - FFFF D3FC$_H$	nE	nE	–
CPR1_0L	Code Seg. Prot. Register Set 1, Range 0, Lower	FFFF D400$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPR1_0U	Code Seg. Prot. Register Set 1, Range 0, Upper	FFFF D404$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPR1_1L	Code Seg. Prot. Register Set 1, Range 1, Lower	FFFF D408$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPR1_1U	Code Seg. Prot. Register Set 1, Range 1, Upper	FFFF D40C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	FFFF D410$_H$ - FFFF DFFC$_H$	nE	nE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
DPM0	Data Memory Protection Mode Register 0	FFFF E000$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	FFFF E004$_H$ - FFFF E07C$_H$	nE	nE	–
DPM1	Data Memory Protection Mode Register 1	FFFF E080$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	FFFF E084$_H$ - FFFF E1FC$_H$	nE	nE	–
CPM0	Code Memory Protection Mode Register 0	FFFF E200$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	FFFF E204$_H$ - FFFF E27C$_H$	nE	nE	–
CPM1	Code Memory Protection Mode Register 1	FFFF E280$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	FFFF E284$_H$ - FFFF EFFF$_H$	nE	nE	–
Core Debug Register (OCDS)					
DBGSR	Debug Status Register	FFFF FD00$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	FFFF FD04$_H$	nE	nE	–
EXEVT	External Break Input Event Specifier Register	FFFF FD08$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CREVT	Emulator Resource Protection Event Specifier Register	FFFF FD0C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
SWEVT	Software Break Event Specifier Register	FFFF FD10$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	FFFF FD14$_H$ - FFFF FD1C$_H$	nBE	nBE	–
TR0EVT	Trigger Event 0 Specifier Register	FFFF FD20$_H$	U, SV, 32	SV, 32	0000 0000$_H$
TR1EVT	Trigger Event 1 Specifier Register	FFFF FD24$_H$	U, SV, 32	SV, 32	0000 0000$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
–	Reserved	FFFF FD28$_H$ - FFFF FDFC$_H$	nBE	nBE	–
Core Special Function Registers (CSFR)					
PCXI	Previous Context Information Register	FFFF FE00$_H$	U, SV, 32	SV, 32	0000 0000$_H$
PSW	Program Status Word	FFFF FE04$_H$	U, SV, 32	SV, 32	0000 0B80$_H$
PC	Program Counter	FFFF FE08$_H$	U, SV, 32	SV, 32	acc. boot cfg.
–	Reserved	FFFF FE0C$_H$ - FFFF FE10$_H$	nBE	nBE	–
SYSCON	System Configuration Register	FFFF FE14$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	FFFF FE18$_H$ - FFFF FE1C$_H$	nBE	nBE	–
BIV	Interrupt Vector Table Pointer	FFFF FE20$_H$	U, SV, 32	SV, E; 32	0000 0000$_H$
BTV	Trap Vector Table Pointer	FFFF FE24$_H$	U, SV, 32	SV, E; 32	A000 0100$_H$
ISP	Interrupt Stack Pointer	FFFF FE28$_H$	U, SV, 32	SV, E; 32	0000 0100$_H$
ICR	ICU Interrupt Control Register	FFFF FE2C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	FFFF FE30$_H$ - FFFF FE34$_H$	nBE	nBE	–
FCX	Free Context List Head Pointer	FFFF FE38$_H$	U, SV, 32	SV, 32	0000 0000$_H$
LCX	Free Context List Limit Pointer	FFFF FE3C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	–	FFFF FE40$_H$ - FFFF FEFC$_H$	nBE	nBE	–

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
General Purpose Register (GPR)					
D0	Data Register D0 (DGPR)	FFFF FF00$_H$	–	–	XXXX XXXX$_H$
D1	Data Register D1 (DGPR)	FFFF FF04$_H$	–	–	XXXX XXXX$_H$
D2	Data Register D2 (DGPR)	FFFF FF08$_H$	–	–	XXXX XXXX$_H$
D3	Data Register D3 (DGPR)	FFFF FF0C$_H$	–	–	XXXX XXXX$_H$
D4	Data Register D4 (DGPR)	FFFF FF10$_H$	–	–	XXXX XXXX$_H$
D5	Data Register D5 (DGPR)	FFFF FF14$_H$	–	–	XXXX XXXX$_H$
D6	Data Register D6 (DGPR)	FFFF FF18$_H$	–	–	XXXX XXXX$_H$
D7	Data Register D7 (DGPR)	FFFF FF1C$_H$	–	–	XXXX XXXX$_H$
D8	Data Register D8 (DGPR)	FFFF FF20$_H$	–	–	XXXX XXXX$_H$
D9	Data Register D9 (DGPR)	FFFF FF24$_H$	–	–	XXXX XXXX$_H$
D10	Data Register 10 (DGPR)	FFFF FF28$_H$	–	–	XXXX XXXX$_H$
D11	Data Register 11 (DGPR)	FFFF FF2C$_H$	–	–	XXXX XXXX$_H$
D12	Data Register 12 (DGPR)	FFFF FF30$_H$	–	–	XXXX XXXX$_H$
D13	Data Register 13 (DGPR)	FFFF FF34$_H$	–	–	XXXX XXXX$_H$
D14	Data Register 14 (DGPR)	FFFF FF38$_H$	–	–	XXXX XXXX$_H$
D15	Data Register 15 (DGPR)	FFFF FF3C$_H$	–	–	XXXX XXXX$_H$
–	Reserved	FFFF FF40$_H$ - FFFF FF7C$_H$	nE	nE	–
A0	Address Reg. 0 (AGPR) Global Address Register	FFFF FF80$_H$	–	–	XXXX XXXX$_H$
A1	Address Reg. 1 (AGPR) Global Address Register	FFFF FF84$_H$	–	–	XXXX XXXX$_H$
A2	Address Register 2 (AGPR)	FFFF FF88$_H$	–	–	XXXX XXXX$_H$
A3	Address Register 3 (AGPR)	FFFF FF8C$_H$	–	–	XXXX XXXX$_H$
A4	Address Register 4 (AGPR)	FFFF FF90$_H$	–	–	XXXX XXXX$_H$
A5	Address Register 5 (AGPR)	FFFF FF94$_H$	–	–	XXXX XXXX$_H$

Table 20-5 Detailed Address Map of Segment 15 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
A6	Address Register 6 (AGPR)	FFFF FF98$_H$	–	–	XXXX XXXX$_H$
A7	Address Register 7 (AGPR)	FFFF FF9C$_H$	–	–	XXXX XXXX$_H$
A8	Address Reg. 8 (AGPR) Global Address Register	FFFF FFA0$_H$	–	–	XXXX XXXX$_H$
A9	Address Reg. 9 (AGPR) Global Address Register	FFFF FFA4$_H$	–	–	XXXX XXXX$_H$
A10 (SP)	Address Reg. 10 (AGPR) Stack Pointer	FFFF FFA8$_H$	–	–	XXXX XXXX$_H$
A11 (RA)	Address Reg. 11 (AGPR) Return Address	FFFF FFAC$_H$	–	–	XXXX XXXX$_H$
A12	Address Reg. 12 (AGPR)	FFFF FFB0$_H$	–	–	XXXX XXXX$_H$
A13	Address Reg. 13 (AGPR)	FFFF FFB4$_H$	–	–	XXXX XXXX$_H$
A14	Address Reg. 14 (AGPR)	FFFF FFB8$_H$	–	–	XXXX XXXX$_H$
A15	Address Reg. 15 (AGPR)	FFFF FFBC$_H$	–	–	XXXX XXXX$_H$
–	Reserved	FFFF FFC0$_H$ - FFFF FFFC$_H$	nE	nE	–

21 Index

21.1 Keyword Index

This section lists a number of keywords which refer to specific details of the TC1765 in terms of its architecture, its functional units, or functions. Bold page number entries identify the main definition material for a topic.

A

Abbreviations 1-4
Address map 7-2, 7-5

B

BCU
- Block diagram 16-4
- Registers 16-10
  - Address range 16-10
  - BCU_CON **16-11**
  - BCU_EADD **16-14**
  - BCU_ECON **16-13**
  - BCU_EDAT **16-14**
  - BCU_SRC **16-15**
  - Offset addresses 16-10
  - Overview 16-10

Boot operation 5-12–5-15
- Boot configuration handling 5-14
- Boot selection table 5-13
- Debug boot options 5-14
- Hardware boot 5-12
- Normal boot options 5-14
- Software boot 5-12

Boot ROM 8-6
Burst mode timings 13-35–13-42

C

Clock gating and power management 3-12
- CLC register implementations 3-20
- Module clock control register 3-15
- Module clock generation 3-14

Clock generation unit 3-3–3-11
- Clock control and status register 3-9
- Oscillator circuit 3-4
- PLL loss and lock 3-10
- PLL operation 3-5
- Setup of system clock frequency 3-7
- Startup operation 3-10

Clock system block diagram 3-2
CPS 2-31
CPU 2-1
- Block diagram 2-2
- Core SFRs
  - Address table **2-32**–**2-33**
  - BIV **2-27**
  - BTV **2-28**
  - FCX **2-23**
  - ISP **2-26**
  - LCX **2-25**
  - PC **2-16**
  - PCX **2-24**
  - PCXI **2-21**
  - PSW **2-17**
- Execution unit 2-4
- Instruction fetch unit 2-3
- Service request nodes 14-22

CPU architecture overview 2-1
- Addressing modes 2-7
- Data types 2-7
- Instruction formats 2-7
- Interrupt system 2-10
- Processor registers 2-13–2-30
  - Context management registers 2-23
  - Debug registers 2-30
  - Interrupt/trap control registers 2-27

Memory protection registers 2-30
Program state registers 2-16
Stack registers 2-26
System control register 2-29
Program state registers 2-6
Protection system 2-11–2-12
Reset system 2-12
Tasks and contexts 2-7
Trap system 2-10
CPU General purpose register file 2-5

D

Data memory unit 9-1–9-17
Address map 9-2
Block diagram 9-1
Bus error 9-3
DMU overly operation 9-5
DMU register access error 9-4
DMU trap generation 9-3
Range error 9-4
Registers 9-10
Address range 9-11
DMU_ATR **9-13**
DMU_CON **9-11**
DMU_EOCRn **9-15**
DMU_IOCRn **9-14**
DMU_IORBAP **9-17**
DMU_STR **9-12**
Offset addresses 9-10
Overview 9-10
DMA controller
Block diagram 10-5
Channel configuration 10-7
Definitions 10-3
DMA transaction 10-3
DMA transfer 10-3
Error conditions 10-13
Features 10-2
General operation 10-4
Hardware controlled continuous mode 10-11
Hardware controlled single mode 10-10
Module implementation 10-28–10-34
External registers 10-31
Interconnections 10-28
Request assignment 10-29
Register interface 10-6
Registers 10-17–10-27
CSRn **10-22**
DEAn **10-27**
DSAn **10-27**
GCTR **10-19**
Offset addresses 10-18
OTCn **10-25**
Overview 10-17
SEAn **10-26**
SSAn **10-26**
Request assignment unit 0/1 10-14
Request assignment unit 2 10-15
Software controlled continuous mode 10-9
Software controlled single mode 10-8
Transfer parameter 10-12
Address calculation 10-12
Transfer count 10-12
DMA, see “DMA controller”
DMU, see “Data memory unit”
Document
Abbreviations 1-4
Structure 1-1
Terminology 1-3
Textual conventions 1-1

E

EBU, see “External bus interface unit”
Endinit function 18-3
External bus interface unit 13-1–13-55
Address region parameters 13-17
Address region selection 13-14
Basic access timing 13-21
Demultiplexed mode 13-21
Basic operation 13-4
Block diagram 13-1
Boot process 13-9

External boot memory configuration word 13-10
Timing 13-11
Data width 13-19
Emulation support 13-27
Emulation boot 13-27
Overlay memory 13-27
Example configuration 13-4
Features 13-3
I/O line characteristics 12-34
Instruction fetches 13-30
Basic functions 13-30
External instruction fetch control register 13-32
Internal to external operation 13-12
Overview 13-2
Pull-up/pull-down control 4-5, 12-34
Registers 13-43
Address range 13-55
EBU_ADDSELx **13-46**
EBU_BUSCONx **13-47**
EBU_CLC **13-44**
EBU_CON **13-45**
EBU_EMUAS **13-54**
EBU_EMUBC **13-51**
EBU_EMUCON **13-50**
Offset addresses 13-43
Overview 13-43
Signal description 13-5
External overlay memory 13-27

F

Features 1-6
CPU 1-6
Development support 1-8
External bus interface 1-7
Instruction set 1-7
Interrupt system 1-7
On-chip memory 1-7
FPI Bus
Arbitration 16-5
Block diagram 16-3
Bus agents 16-5
Error handling 16-6
Overview 16-1
Power saving mode 16-9
Starvation protection 16-6

I

Instruction cache 8-3
Instruction set overview 2-34
Arithmetic comparison 2-53
Arithmetic instructions 2-34–2-45
Branch instructions 2-54–2-56
Compare instructions 2-46–2-52
Context related instructions 2-60
DSP arithmetic 2-42
Load/store instructions 2-57–2-60
System instructions 2-61
Interrupt system 14-1–14-26
Arbitration cycles 14-11
Arbitration process 14-11
Block diagram 14-2
External interrupts 14-3
Hints for applications 14-17–14-21
Interrupt control unit 14-7
Interrupt vector table 14-14
Overview 14-1
Service request control register 14-3
Service request nodes 14-3
Service routine entering 14-12
Service routine exiting 14-13

M

Memories, see On-chip memories
Memory map 7-1
Memory protection system 11-1–11-19
Configuration example 11-17
Memory access checking 11-18
Overview 11-1
Registers
Address Range 11-6
Control by PSW bits/bit fields 11-7
for code memory protection 11-14
for data memory protection 11-11
Offset addresses 11-4

Overview 11-2

N

NMI 15-12
NMI input 15-13
PLL NMI 15-13
Status register NMISR 15-12
Watchdog timer NMI 15-14

O

OCDS, see “On-chip debug support”
On-chip debug support 19-1–19-33
Block diagram 19-1
Registers 19-8
Address ranges 19-33
CREVT **19-12**
DBGSR **19-9**
EXEVT **19-11**
IOCONF **19-29**
IOSR **19-31**
Offset addresses 19-8
Overview 19-8
SBSRC0 **19-15**
SWEVT **19-13**
TR0EVT **19-14**
TR1EVT **19-14**
Trace module 19-17
TriCore CPU debugging 19-2
BRKOUT pin 19-6
External debug event 19-3
Instruction debug event 19-3
Protection violation triggers 19-4
Software debug event 19-3
On-chip memories 7-1–7-6
Address map of segment 15 7-5
General address map 7-2
Memory map 7-1

P

Parallel ports
Block diagram 12-1
General operation 12-1
General port structure 12-3
Kernel registers 12-4
Alternate function select register 12-14
Data input register 12-7
Data output register 12-6
Direction control register 12-8
Input configuration register 12-10
Offset addresses 12-4
Open drain control register 12-9
Output driver characteristics register 12-13
Pull-up/-down control registers 12-11
Px_ALTSELn **12-14**
Px_DIR **12-8**
Px_IN **12-7**
Px_OD **12-9**
Px_OUT **12-6**
Px_PICON **12-10**
Px_POCON **12-13**
Px_PUDEN **12-12**
Px_PUDSEL **12-11**
Registers 12-4
PMU, see “Program memory unit”
Power Management 6-1–6-10
Mode description 6-6
Deep sleep mode 6-7
Idle mode 6-6
Sleep mode 6-6
Mode summary 6-1, 6-10
Overview 6-1
Registers 6-3
Address range 6-3
Offset addresses 6-3
Overview 6-3
PMG_CON 6-4
PMG_CSR 6-5
Program memory unit 8-1
Block diagram 8-1
Boot ROM 8-6
External instruction fetch example 8-6
External instruction fetches 8-5
Instruction cache 8-3

Registers 8-7
Offset addresses 8-7
Overview 8-7
PMU_CON **8-8**
PMU_EIFCON **8-10**, **13-32**
Related memories 8-2
Scratch-pad code RAM 8-3

R

Reset operation 5-1–5-11
Deep sleep wake-up reset 5-10
External hardware reset 5-7
Overview 5-1
Power-on reset 5-7
Registers
Address range 5-2
Offset addresses 5-2
Overview 5-2
RST_REQ **5-5**
RST_SR **5-3**
Reset register table 5-9
Software reset 5-8
States after reset 5-10
Watchdog timer reset 5-9

S

Scratch-pad code RAM 8-3
SCU, see “System control unit”
STM, see “System timer”
System control unit 4-1–4-11
Address range 4-2
Overview 4-1
Port 5 trace control 4-7
Registers
CHIPID **4-10**
MANID **4-9**
Offset addresses 4-2
Overview 4-2
RTID **4-11**
SCU_CON **4-3**
SCU_TRSTAT **4-8**
Trace status 4-8
System timer
Block diagram 17-2
Overview 17-1
Registers
Address range 17-8
CAP **17-6**
Offset addresses 17-4
Overview 17-4
TIM0 **17-5**
TIM1 **17-5**
TIM2 **17-5**
TIM3 **17-5**
TIM4 **17-6**
TIM5 **17-6**
TIM6 **17-6**
Resolutions and ranges 17-3

T

Trap system 15-1–15-14
Asynchronous traps 15-5
Hardware traps 15-5
Overview 15-1
Service routine 15-11
Software traps 15-5
Synchronous traps 15-5
Trap classes 15-3
Trap descriptions 15-6
Trap vector table 15-10

W

Watchdog timer 18-1–18-34
Double watchdog error 18-17
During power-saving modes 18-17
Endinit function 18-3
Features 18-2
Functional description 18-5
in OCDS suspend mode 18-17
Modes of operation 18-7
Disable mode 18-8, 18-15
Normal mode 18-8, 18-14
Prewarning mode 18-9, 18-16
Time-out mode 18-8, 18-13
Modify access to WDT_CON0 18-11
Monitoring diagram 18-26

Operation sequence example 18-5
Overview 18-1
Period calculation 18-18
Period in power-saving modes 18-21
Registers 18-28
Offset addresses 18-28
WDT_CON0 **18-29**
WDT_CON1 **18-31**
WDT_SR **18-32**
Service sequence diagram 18-25
Servicing 18-23
System initialization 18-22
Time-out period 18-19
Watchdog timer reset lock 5-9
WDT, see “Watchdog timer”

21.2 Register Index

This section lists the references to the Special Function Registers of the TC1765.

A

A0 2-32, 20-83
A1 2-33, 20-83
A10 2-33, 20-84
A11 2-33, 20-84
A12 2-33, 20-84
A13 2-33, 20-84
A14 2-33, 20-84
A15 2-33, 20-84
A2 2-33, 20-83
A3 2-33, 20-83
A4 2-33, 20-83
A5 2-33, 20-83
A6 2-33, 20-84
A7 2-33, 20-84
A8 2-33, 20-84
A9 2-33, 20-84
ADC0_AP 20-43
ADC0_ASCRP 20-45
ADC0_CHCON0 20-41
ADC0_CHCON1 20-41
ADC0_CHCON10 20-42
ADC0_CHCON11 20-42
ADC0_CHCON12 20-42
ADC0_CHCON13 20-42
ADC0_CHCON14 20-42
ADC0_CHCON15 20-42
ADC0_CHCON2 20-41
ADC0_CHCON3 20-41
ADC0_CHCON4 20-41
ADC0_CHCON5 20-42
ADC0_CHCON6 20-42
ADC0_CHCON7 20-42
ADC0_CHCON8 20-42
ADC0_CHCON9 20-42
ADC0_CHIN 20-43
ADC0_CHSTAT0 20-44
ADC0_CHSTAT1 20-44
ADC0_CHSTAT10 20-44
ADC0_CHSTAT11 20-44
ADC0_CHSTAT12 20-44
ADC0_CHSTAT13 20-44
ADC0_CHSTAT14 20-45
ADC0_CHSTAT15 20-45
ADC0_CHSTAT2 20-44
ADC0_CHSTAT3 20-44
ADC0_CHSTAT4 20-44
ADC0_CHSTAT5 20-44
ADC0_CHSTAT6 20-44
ADC0_CHSTAT7 20-44
ADC0_CHSTAT8 20-44
ADC0_CHSTAT9 20-44
ADC0_CLC 20-41
ADC0_CON 20-43
ADC0_EXCRP 20-45
ADC0_EXEV 20-42
ADC0_EXEVC 20-42
ADC0_EXTC0 20-43
ADC0_EXTC1 20-43
ADC0_ID 20-41
ADC0_LCCON0 20-43
ADC0_LCCON1 20-43
ADC0_LCCON2 20-43
ADC0_LCCON3 20-43
ADC0_MSS0 20-46
ADC0_MSS1 20-46
ADC0_QEV 20-42
ADC0_QR 20-43
ADC0_QUEUE0 20-45
ADC0_REQ0 20-44
ADC0_SAL 20-43
ADC0_SCN 20-44
ADC0_SRC0 20-46
ADC0_SRC1 20-46

ADC0_SRC2 20-46
ADC0_SRC3 20-46
ADC0_SRNP 20-46
ADC0_STAT 20-45
ADC0_SW0CRP 20-45
ADC0_SYSTAT 20-45
ADC0_TCON 20-43
ADC0_TCRP 20-45
ADC0_TEV 20-42
ADC0_TSTAT 20-45
ADC0_TTC 20-43
ADC1_AP 20-48
ADC1_ASCRP 20-50
ADC1_CHCON0 20-46
ADC1_CHCON1 20-46
ADC1_CHCON10 20-47
ADC1_CHCON11 20-47
ADC1_CHCON12 20-47
ADC1_CHCON13 20-47
ADC1_CHCON14 20-47
ADC1_CHCON15 20-47
ADC1_CHCON2 20-46
ADC1_CHCON3 20-46
ADC1_CHCON4 20-47
ADC1_CHCON5 20-47
ADC1_CHCON6 20-47
ADC1_CHCON7 20-47
ADC1_CHCON8 20-47
ADC1_CHCON9 20-47
ADC1_CHIN 20-48
ADC1_CHSTAT0 20-49
ADC1_CHSTAT1 20-49
ADC1_CHSTAT10 20-49
ADC1_CHSTAT11 20-49
ADC1_CHSTAT12 20-49
ADC1_CHSTAT13 20-50
ADC1_CHSTAT14 20-50
ADC1_CHSTAT15 20-50
ADC1_CHSTAT2 20-49
ADC1_CHSTAT3 20-49
ADC1_CHSTAT4 20-49
ADC1_CHSTAT5 20-49
ADC1_CHSTAT6 20-49
ADC1_CHSTAT7 20-49
ADC1_CHSTAT8 20-49
ADC1_CHSTAT9 20-49
ADC1_CON 20-49
ADC1_EXCRP 20-51
ADC1_EXEV 20-47
ADC1_EXEVC 20-48
ADC1_EXTC0 20-48
ADC1_EXTC1 20-48
ADC1_LCCON0 20-48
ADC1_LCCON1 20-48
ADC1_LCCON2 20-48
ADC1_LCCON3 20-48
ADC1_MSS0 20-51
ADC1_MSS1 20-51
ADC1_QEV 20-47
ADC1_QR 20-48
ADC1_QUEUE0 20-50
ADC1_REQ0 20-49
ADC1_SAL 20-48
ADC1_SCN 20-49
ADC1_SRC0 20-51
ADC1_SRC1 20-51
ADC1_SRC2 20-51
ADC1_SRC3 20-51
ADC1_SRNP 20-51
ADC1_STAT 20-50
ADC1_SW0CRP 20-50
ADC1_SYSTAT 20-50
ADC1_TCON 20-48
ADC1_TCRP 20-50
ADC1_TEV 20-47
ADC1_TSTAT 20-50
ADC1_TTC 20-48
ASC0_BG 20-17
ASC0_CLC 20-17
ASC0_CON 20-17
ASC0_ESRC 20-18
ASC0_FDV 20-17
ASC0_ID 20-17
ASC0_RBUF 20-18
ASC0_RSRC 20-18
ASC0_TBSRC 20-18

ASC0_TBUF 20-17
ASC0_TSRC 20-18
ASC1_BG 20-18
ASC1_CLC 20-18
ASC1_CON 20-18
ASC1_ESRC 20-19
ASC1_FDV 20-18
ASC1_ID 20-18
ASC1_RBUF 20-19
ASC1_RSRC 20-19
ASC1_TBSRC 20-19
ASC1_TBUF 20-18
ASC1_TSRC 20-19

B

BCU module registers 16-10
BCU_CON 16-11, 20-12
BCU_EADD 16-14, 20-12
BCU_ECON 16-13, 20-12
BCU_EDAT 16-14, 20-12
BCU_ID 20-12
BCU_SRC 16-15, 20-12
BIV 2-27, 2-32, 20-82
BTV 2-28, 2-32, 20-82

C

CAN_ABTR 20-60
CAN_ACR 20-60
CAN_AECNT 20-61
CAN_AFCR 20-60
CAN_AGINP 20-60
CAN_AIMR0 20-61
CAN_AIMR4 20-61
CAN_AIR 20-60
CAN_ASR 20-60
CAN_BBTR 20-61
CAN_BCR 20-61
CAN_BECNT 20-61
CAN_BFCR 20-61
CAN_BGINP 20-61
CAN_BIMR0 20-61
CAN_BIMR4 20-61
CAN_BIR 20-61
CAN_BSR 20-61
CAN_CLC 20-60
CAN_ID 20-60
CAN_MSGAMR0 20-62
CAN_MSGAMR1 20-62
CAN_MSGAMR10 20-67
CAN_MSGAMR11 20-67
CAN_MSGAMR12 20-68
CAN_MSGAMR13 20-68
CAN_MSGAMR14 20-69
CAN_MSGAMR15 20-69
CAN_MSGAMR16 20-70
CAN_MSGAMR17 20-70
CAN_MSGAMR18 20-71
CAN_MSGAMR19 20-71
CAN_MSGAMR2 20-63
CAN_MSGAMR20 20-72
CAN_MSGAMR21 20-72
CAN_MSGAMR22 20-73
CAN_MSGAMR23 20-73
CAN_MSGAMR24 20-74
CAN_MSGAMR25 20-74
CAN_MSGAMR26 20-75
CAN_MSGAMR27 20-75
CAN_MSGAMR28 20-76
CAN_MSGAMR29 20-76
CAN_MSGAMR3 20-63
CAN_MSGAMR30 20-77
CAN_MSGAMR31 20-77
CAN_MSGAMR4 20-64
CAN_MSGAMR5 20-64
CAN_MSGAMR6 20-65
CAN_MSGAMR7 20-65
CAN_MSGAMR8 20-66
CAN_MSGAMR9 20-66
CAN_MSGAR0 20-62
CAN_MSGAR1 20-62
CAN_MSGAR10 20-67
CAN_MSGAR11 20-67
CAN_MSGAR12 20-68
CAN_MSGAR13 20-68
CAN_MSGAR14 20-69
CAN_MSGAR15 20-69

CAN_MSGAR16 20-70
CAN_MSGAR17 20-70
CAN_MSGAR18 20-71
CAN_MSGAR19 20-71
CAN_MSGAR2 20-63
CAN_MSGAR20 20-72
CAN_MSGAR21 20-72
CAN_MSGAR22 20-73
CAN_MSGAR23 20-73
CAN_MSGAR24 20-74
CAN_MSGAR25 20-74
CAN_MSGAR26 20-75
CAN_MSGAR27 20-75
CAN_MSGAR28 20-76
CAN_MSGAR29 20-76
CAN_MSGAR3 20-63
CAN_MSGAR30 20-77
CAN_MSGAR31 20-77
CAN_MSGAR4 20-64
CAN_MSGAR5 20-64
CAN_MSGAR6 20-65
CAN_MSGAR7 20-65
CAN_MSGAR8 20-66
CAN_MSGAR9 20-66
CAN_MSGCFG0 20-62
CAN_MSGCFG1 20-62
CAN_MSGCFG10 20-67
CAN_MSGCFG11 20-67
CAN_MSGCFG12 20-68
CAN_MSGCFG13 20-68
CAN_MSGCFG14 20-69
CAN_MSGCFG15 20-69
CAN_MSGCFG16 20-70
CAN_MSGCFG17 20-70
CAN_MSGCFG18 20-71
CAN_MSGCFG19 20-71
CAN_MSGCFG2 20-63
CAN_MSGCFG20 20-72
CAN_MSGCFG21 20-72
CAN_MSGCFG22 20-73
CAN_MSGCFG23 20-73
CAN_MSGCFG24 20-74
CAN_MSGCFG25 20-74
CAN_MSGCFG26 20-75
CAN_MSGCFG27 20-75
CAN_MSGCFG28 20-76
CAN_MSGCFG29 20-76
CAN_MSGCFG3 20-63
CAN_MSGCFG30 20-77
CAN_MSGCFG31 20-77
CAN_MSGCFG4 20-64
CAN_MSGCFG5 20-64
CAN_MSGCFG6 20-65
CAN_MSGCFG7 20-65
CAN_MSGCFG8 20-66
CAN_MSGCFG9 20-66
CAN_MSGCTR0 20-62
CAN_MSGCTR1 20-62
CAN_MSGCTR10 20-67
CAN_MSGCTR11 20-67
CAN_MSGCTR12 20-68
CAN_MSGCTR13 20-68
CAN_MSGCTR14 20-69
CAN_MSGCTR15 20-69
CAN_MSGCTR16 20-70
CAN_MSGCTR17 20-70
CAN_MSGCTR18 20-71
CAN_MSGCTR19 20-71
CAN_MSGCTR2 20-63
CAN_MSGCTR20 20-72
CAN_MSGCTR21 20-72
CAN_MSGCTR22 20-73
CAN_MSGCTR23 20-73
CAN_MSGCTR24 20-74
CAN_MSGCTR25 20-74
CAN_MSGCTR26 20-75
CAN_MSGCTR27 20-75
CAN_MSGCTR28 20-76
CAN_MSGCTR29 20-76
CAN_MSGCTR3 20-63
CAN_MSGCTR30 20-77
CAN_MSGCTR31 20-77
CAN_MSGCTR4 20-64
CAN_MSGCTR5 20-64
CAN_MSGCTR6 20-65
CAN_MSGCTR7 20-65

CAN_MSGCTR8 20-66
CAN_MSGCTR9 20-66
CAN_MSGDR00 20-62
CAN_MSGDR04 20-62
CAN_MSGDR10 20-62
CAN_MSGDR100 20-67
CAN_MSGDR104 20-67
CAN_MSGDR110 20-67
CAN_MSGDR114 20-67
CAN_MSGDR120 20-68
CAN_MSGDR124 20-68
CAN_MSGDR130 20-68
CAN_MSGDR134 20-68
CAN_MSGDR14 20-62
CAN_MSGDR140 20-69
CAN_MSGDR144 20-69
CAN_MSGDR150 20-69
CAN_MSGDR154 20-69
CAN_MSGDR160 20-70
CAN_MSGDR164 20-70
CAN_MSGDR170 20-70
CAN_MSGDR174 20-70
CAN_MSGDR180 20-71
CAN_MSGDR184 20-71
CAN_MSGDR190 20-71
CAN_MSGDR194 20-71
CAN_MSGDR20 20-63
CAN_MSGDR200 20-72
CAN_MSGDR204 20-72
CAN_MSGDR210 20-72
CAN_MSGDR214 20-72
CAN_MSGDR220 20-73
CAN_MSGDR224 20-73
CAN_MSGDR230 20-73
CAN_MSGDR234 20-73
CAN_MSGDR24 20-63
CAN_MSGDR240 20-74
CAN_MSGDR244 20-74
CAN_MSGDR250 20-74
CAN_MSGDR254 20-74
CAN_MSGDR260 20-75
CAN_MSGDR264 20-75
CAN_MSGDR270 20-75
CAN_MSGDR274 20-75
CAN_MSGDR280 20-76
CAN_MSGDR284 20-76
CAN_MSGDR290 20-76
CAN_MSGDR294 20-76
CAN_MSGDR30 20-63
CAN_MSGDR300 20-77
CAN_MSGDR304 20-77
CAN_MSGDR310 20-77
CAN_MSGDR314 20-77
CAN_MSGDR34 20-63
CAN_MSGDR40 20-64
CAN_MSGDR44 20-64
CAN_MSGDR50 20-64
CAN_MSGDR54 20-64
CAN_MSGDR60 20-65
CAN_MSGDR64 20-65
CAN_MSGDR70 20-65
CAN_MSGDR74 20-65
CAN_MSGDR80 20-66
CAN_MSGDR84 20-66
CAN_MSGDR90 20-66
CAN_MSGDR94 20-66
CAN_MSGFCR0 20-62
CAN_MSGFCR1 20-62
CAN_MSGFCR10 20-67
CAN_MSGFCR11 20-67
CAN_MSGFCR12 20-68
CAN_MSGFCR13 20-68
CAN_MSGFCR14 20-69
CAN_MSGFCR15 20-69
CAN_MSGFCR16 20-70
CAN_MSGFCR17 20-70
CAN_MSGFCR18 20-71
CAN_MSGFCR19 20-71
CAN_MSGFCR2 20-63
CAN_MSGFCR20 20-72
CAN_MSGFCR21 20-72
CAN_MSGFCR22 20-73
CAN_MSGFCR23 20-73
CAN_MSGFCR24 20-74
CAN_MSGFCR25 20-74
CAN_MSGFCR26 20-75

CAN_MSGFCR27 20-75
CAN_MSGFCR28 20-76
CAN_MSGFCR29 20-76
CAN_MSGFCR3 20-63
CAN_MSGFCR30 20-77
CAN_MSGFCR31 20-77
CAN_MSGFCR4 20-64
CAN_MSGFCR5 20-64
CAN_MSGFCR6 20-65
CAN_MSGFCR7 20-65
CAN_MSGFCR8 20-66
CAN_MSGFCR9 20-66
CAN_RXIPND 20-61
CAN_SRC0 20-78
CAN_SRC1 20-78
CAN_SRC2 20-78
CAN_SRC3 20-78
CAN_SRC4 20-78
CAN_SRC5 20-78
CAN_SRC6 20-78
CAN_SRC7 20-78
CAN_TXIPND 20-61
CHIPID 4-10, 20-12
COMDATA 20-13
CPM0 11-15, 20-81
CPM1 11-15, 20-81
CPR0_0L 11-14, 20-80
CPR0_0U 11-14, 20-80
CPR0_1L 11-14, 20-80
CPR0_1U 11-14, 20-80
CPR1_0L 11-14, 20-80
CPR1_0U 11-14, 20-80
CPR1_1L 11-14, 20-80
CPR1_1U 11-14, 20-80
CPU_ID 20-78
CPU_SRC0 14-22, 20-79
CPU_SRC1 14-22, 20-79
CPU_SRC2 14-22, 20-79
CPU_SRC3 14-22, 20-79
CREVT 19-12, 20-81

D

D0 2-32, 20-83
D1 2-32, 20-83
D10 2-32, 20-83
D11 2-32, 20-83
D12 2-32, 20-83
D13 2-32, 20-83
D14 2-32, 20-83
D15 2-32, 20-83
D2 2-32, 20-83
D3 2-32, 20-83
D4 2-32, 20-83
D5 2-32, 20-83
D6 2-32, 20-83
D7 2-32, 20-83
D8 2-32, 20-83
D9 2-32, 20-83
DBGSR 19-9, 20-81
DEA03 20-58, 20-59
DMA_CLC 10-32, 20-56
DMA_CSR00 20-56
DMA_CSR01 20-56
DMA_CSR02 20-57
DMA_CSR03 20-57
DMA_CSR10 20-58
DMA_CSR11 20-58
DMA_CSR12 20-59
DMA_CSR13 20-59
DMA_CSRn 10-22
DMA_DEA00 20-56
DMA_DEA01 20-57
DMA_DEA02 20-57
DMA_DEA03 20-58
DMA_DEA10 20-58
DMA_DEA11 20-58
DMA_DEA12 20-59
DMA_DEA13 20-59
DMA_DEAn 10-27
DMA_DSA00 20-56
DMA_DSA01 20-57
DMA_DSA02 20-57
DMA_DSA03 20-58
DMA_DSA10 20-58
DMA_DSA11 20-58
DMA_DSA12 20-59

DMA_DSA13 20-59
DMA_DSAn 10-27
DMA_GCTR 10-19, 20-56
DMA_ID 20-56
DMA_OTC00 20-56
DMA_OTC01 20-56
DMA_OTC02 20-57
DMA_OTC03 20-57
DMA_OTC10 20-58
DMA_OTC11 20-58
DMA_OTC12 20-59
DMA_OTC13 20-59
DMA_OTCn 10-25
DMA_SEA00 20-56
DMA_SEA01 20-57
DMA_SEA02 20-57
DMA_SEA03 20-57
DMA_SEA10 20-58
DMA_SEA11 20-58
DMA_SEA12 20-59
DMA_SEA13 20-59
DMA_SEAn 10-26
DMA_SRC0 10-33, 20-60
DMA_SRC1 10-33, 20-60
DMA_SRC2 10-33, 20-60
DMA_SRC3 10-33, 20-60
DMA_SRC4 10-33, 20-60
DMA_SRC5 10-33, 20-60
DMA_SRC6 10-33, 20-60
DMA_SRC7 10-33, 20-59
DMA_SSA00 20-56
DMA_SSA01 20-57
DMA_SSA02 20-57
DMA_SSA03 20-57
DMA_SSA10 20-58
DMA_SSA11 20-58
DMA_SSA12 20-59
DMA_SSA13 20-59
DMA_SSAn 10-26
DMU module registers 9-10
DMU_ATR 9-13, 20-4
DMU_CON 9-11, 20-4
DMU_EOCR0 9-15, 20-6
DMU_EOCR1 9-15, 20-6
DMU_ID 20-4
DMU_IOCR0 9-14, 20-5
DMU_IOCR1 9-14, 20-5
DMU_IOCR2 9-14, 20-5
DMU_IOCR3 9-14, 20-6
DMU_IORBAP 9-17, 20-6
DMU_STR 9-12, 20-4
DPM0 11-12, 20-81
DPM1 11-12, 20-81
DPR0_0L 11-11, 20-79
DPR0_0U 11-11, 20-79
DPR0_1L 11-11, 20-79
DPR0_1U 11-11, 20-79
DPR0_2L 11-11, 20-79
DPR0_2U 11-11, 20-79
DPR0_3L 11-11, 20-79
DPR0_3U 11-11, 20-79
DPR1_0L 11-11, 20-79
DPR1_0U 11-11, 20-79
DPR1_1L 11-11, 20-79
DPR1_1U 11-11, 20-80
DPR1_2L 11-11, 20-80
DPR1_2U 11-11, 20-80
DPR1_3L 11-11, 20-80
DPR1_3U 11-11, 20-80

E

EBU module registers 13-43
EBU_ADDSEL0 13-46, 20-14
EBU_ADDSEL1 13-46, 20-14
EBU_ADDSEL2 13-46, 20-14
EBU_ADDSEL3 13-46, 20-14
EBU_BUSCON0 13-47, 20-14
EBU_BUSCON1 13-47, 20-14
EBU_BUSCON2 13-47, 20-14
EBU_BUSCON3 13-47, 20-14
EBU_CLC 13-44, 20-14
EBU_CON 13-45, 20-14
EBU_EMUAS 13-54, 20-14
EBU_EMUBC 13-51, 20-15
EBU_EMUCON 13-50, 20-15
EBU_ID 20-14

EXEVT 19-11, 20-81

F

FCX 2-23, 2-32, 20-82

G

GPTA_ADCCTR 20-22
GPTA_CKBCTR 20-25
GPTA_CLC 20-21
GPTA_DBGCC 20-22
GPTA_DCMCAV0 20-23
GPTA_DCMCAV1 20-24
GPTA_DCMCAV2 20-24
GPTA_DCMCAV3 20-25
GPTA_DCMCOV0 20-24
GPTA_DCMCOV1 20-24
GPTA_DCMCOV2 20-24
GPTA_DCMCOV3 20-25
GPTA_DCMCTR0 20-23
GPTA_DCMCTR1 20-24
GPTA_DCMCTR2 20-24
GPTA_DCMCTR3 20-24
GPTA_DCMTIM0 20-23
GPTA_DCMTIM1 20-24
GPTA_DCMTIM2 20-24
GPTA_DCMTIM3 20-24
GPTA_EMGCTR0 20-22
GPTA_EMGCTR1 20-22
GPTA_FPCCOM0 20-22
GPTA_FPCCOM1 20-23
GPTA_FPCCOM2 20-23
GPTA_FPCCOM3 20-23
GPTA_FPCCOM4 20-23
GPTA_FPCCOM5 20-23
GPTA_FPCCTR1 20-22
GPTA_FPCCTR2 20-22
GPTA_FPCTIM0 20-22
GPTA_FPCTIM1 20-23
GPTA_FPCTIM2 20-23
GPTA_FPCTIM3 20-23
GPTA_FPCTIM4 20-23
GPTA_FPCTIM5 20-23
GPTA_GTCCTR00 20-26
GPTA_GTCCTR01 20-26
GPTA_GTCCTR02 20-26
GPTA_GTCCTR03 20-26
GPTA_GTCCTR04 20-26
GPTA_GTCCTR05 20-26
GPTA_GTCCTR06 20-26
GPTA_GTCCTR07 20-26
GPTA_GTCCTR08 20-27
GPTA_GTCCTR09 20-27
GPTA_GTCCTR10 20-27
GPTA_GTCCTR11 20-27
GPTA_GTCCTR12 20-27
GPTA_GTCCTR13 20-27
GPTA_GTCCTR14 20-27
GPTA_GTCCTR15 20-27
GPTA_GTCCTR16 20-28
GPTA_GTCCTR17 20-28
GPTA_GTCCTR18 20-28
GPTA_GTCCTR19 20-28
GPTA_GTCCTR20 20-28
GPTA_GTCCTR21 20-28
GPTA_GTCCTR22 20-28
GPTA_GTCCTR23 20-28
GPTA_GTCCTR24 20-29
GPTA_GTCCTR25 20-29
GPTA_GTCCTR26 20-29
GPTA_GTCCTR27 20-29
GPTA_GTCCTR28 20-29
GPTA_GTCCTR29 20-29
GPTA_GTCCTR30 20-29
GPTA_GTCCTR31 20-29
GPTA_GTCTR0 20-25
GPTA_GTCTR1 20-25
GPTA_GTCXR00 20-26
GPTA_GTCXR01 20-26
GPTA_GTCXR02 20-26
GPTA_GTCXR03 20-26
GPTA_GTCXR04 20-26
GPTA_GTCXR05 20-26
GPTA_GTCXR06 20-26
GPTA_GTCXR07 20-26
GPTA_GTCXR08 20-27
GPTA_GTCXR09 20-27

GPTA_GTCXR10 20-27
GPTA_GTCXR11 20-27
GPTA_GTCXR12 20-27
GPTA_GTCXR13 20-27
GPTA_GTCXR14 20-27
GPTA_GTCXR15 20-27
GPTA_GTCXR16 20-28
GPTA_GTCXR17 20-28
GPTA_GTCXR18 20-28
GPTA_GTCXR19 20-28
GPTA_GTCXR20 20-28
GPTA_GTCXR21 20-28
GPTA_GTCXR22 20-28
GPTA_GTCXR23 20-28
GPTA_GTCXR24 20-29
GPTA_GTCXR25 20-29
GPTA_GTCXR26 20-29
GPTA_GTCXR27 20-29
GPTA_GTCXR28 20-29
GPTA_GTCXR29 20-29
GPTA_GTCXR30 20-29
GPTA_GTCXR31 20-29
GPTA_GTREV0 20-25
GPTA_GTREV1 20-25
GPTA_GTTIM0 20-25
GPTA_GTTIM1 20-25
GPTA_ID 20-21
GPTA_LTCCTR00 20-30
GPTA_LTCCTR01 20-30
GPTA_LTCCTR02 20-30
GPTA_LTCCTR03 20-30
GPTA_LTCCTR04 20-30
GPTA_LTCCTR05 20-30
GPTA_LTCCTR06 20-30
GPTA_LTCCTR07 20-30
GPTA_LTCCTR08 20-31
GPTA_LTCCTR09 20-31
GPTA_LTCCTR10 20-31
GPTA_LTCCTR11 20-31
GPTA_LTCCTR12 20-31
GPTA_LTCCTR13 20-31
GPTA_LTCCTR14 20-31
GPTA_LTCCTR15 20-31
GPTA_LTCCTR16 20-32
GPTA_LTCCTR17 20-32
GPTA_LTCCTR18 20-32
GPTA_LTCCTR19 20-32
GPTA_LTCCTR20 20-32
GPTA_LTCCTR21 20-32
GPTA_LTCCTR22 20-32
GPTA_LTCCTR23 20-32
GPTA_LTCCTR24 20-33
GPTA_LTCCTR25 20-33
GPTA_LTCCTR26 20-33
GPTA_LTCCTR27 20-33
GPTA_LTCCTR28 20-33
GPTA_LTCCTR29 20-33
GPTA_LTCCTR30 20-33
GPTA_LTCCTR31 20-33
GPTA_LTCCTR32 20-34
GPTA_LTCCTR33 20-34
GPTA_LTCCTR34 20-34
GPTA_LTCCTR35 20-34
GPTA_LTCCTR36 20-34
GPTA_LTCCTR37 20-34
GPTA_LTCCTR38 20-34
GPTA_LTCCTR39 20-34
GPTA_LTCCTR40 20-35
GPTA_LTCCTR41 20-35
GPTA_LTCCTR42 20-35
GPTA_LTCCTR43 20-35
GPTA_LTCCTR44 20-35
GPTA_LTCCTR45 20-35
GPTA_LTCCTR46 20-35
GPTA_LTCCTR47 20-35
GPTA_LTCCTR48 20-36
GPTA_LTCCTR49 20-36
GPTA_LTCCTR50 20-36
GPTA_LTCCTR51 20-36
GPTA_LTCCTR52 20-36
GPTA_LTCCTR53 20-36
GPTA_LTCCTR54 20-36
GPTA_LTCCTR55 20-36
GPTA_LTCCTR56 20-37
GPTA_LTCCTR57 20-37
GPTA_LTCCTR58 20-37

GPTA_LTCCTR59 20-37
GPTA_LTCCTR60 20-37
GPTA_LTCCTR61 20-37
GPTA_LTCCTR62 20-37
GPTA_LTCCTR63 20-37
GPTA_LTCXR00 20-30
GPTA_LTCXR01 20-30
GPTA_LTCXR02 20-30
GPTA_LTCXR03 20-30
GPTA_LTCXR04 20-30
GPTA_LTCXR05 20-30
GPTA_LTCXR06 20-30
GPTA_LTCXR07 20-30
GPTA_LTCXR08 20-31
GPTA_LTCXR09 20-31
GPTA_LTCXR10 20-31
GPTA_LTCXR11 20-31
GPTA_LTCXR12 20-31
GPTA_LTCXR13 20-31
GPTA_LTCXR14 20-31
GPTA_LTCXR15 20-31
GPTA_LTCXR16 20-32
GPTA_LTCXR17 20-32
GPTA_LTCXR18 20-32
GPTA_LTCXR19 20-32
GPTA_LTCXR20 20-32
GPTA_LTCXR21 20-32
GPTA_LTCXR22 20-32
GPTA_LTCXR23 20-32
GPTA_LTCXR24 20-33
GPTA_LTCXR25 20-33
GPTA_LTCXR26 20-33
GPTA_LTCXR27 20-33
GPTA_LTCXR28 20-33
GPTA_LTCXR29 20-33
GPTA_LTCXR30 20-33
GPTA_LTCXR31 20-33
GPTA_LTCXR32 20-34
GPTA_LTCXR33 20-34
GPTA_LTCXR34 20-34
GPTA_LTCXR35 20-34
GPTA_LTCXR36 20-34
GPTA_LTCXR37 20-34
GPTA_LTCXR38 20-34
GPTA_LTCXR39 20-34
GPTA_LTCXR40 20-35
GPTA_LTCXR41 20-35
GPTA_LTCXR42 20-35
GPTA_LTCXR43 20-35
GPTA_LTCXR44 20-35
GPTA_LTCXR45 20-35
GPTA_LTCXR46 20-35
GPTA_LTCXR47 20-35
GPTA_LTCXR48 20-36
GPTA_LTCXR49 20-36
GPTA_LTCXR50 20-36
GPTA_LTCXR51 20-36
GPTA_LTCXR52 20-36
GPTA_LTCXR53 20-36
GPTA_LTCXR54 20-36
GPTA_LTCXR55 20-36
GPTA_LTCXR56 20-37
GPTA_LTCXR57 20-37
GPTA_LTCXR58 20-37
GPTA_LTCXR59 20-37
GPTA_LTCXR60 20-37
GPTA_LTCXR61 20-37
GPTA_LTCXR62 20-37
GPTA_LTCXR63 20-37
GPTA_MRACTL 20-22
GPTA_MRADIN 20-22
GPTA_MRADOUT 20-22
GPTA_PDLCTR 20-23
GPTA_PLLCNT 20-25
GPTA_PLLCTR 20-25
GPTA_PLLDTR 20-25
GPTA_PLLMTI 20-25
GPTA_PLLREV 20-25
GPTA_PLLSTP 20-25
GPTA_SRC00 20-41
GPTA_SRC01 20-41
GPTA_SRC02 20-41
GPTA_SRC03 20-41
GPTA_SRC04 20-41
GPTA_SRC05 20-41
GPTA_SRC06 20-41

GPTA_SRC07 20-40
GPTA_SRC08 20-40
GPTA_SRC09 20-40
GPTA_SRC10 20-40
GPTA_SRC11 20-40
GPTA_SRC12 20-40
GPTA_SRC13 20-40
GPTA_SRC14 20-40
GPTA_SRC15 20-40
GPTA_SRC16 20-40
GPTA_SRC17 20-40
GPTA_SRC18 20-40
GPTA_SRC19 20-40
GPTA_SRC20 20-40
GPTA_SRC21 20-40
GPTA_SRC22 20-40
GPTA_SRC23 20-39
GPTA_SRC24 20-39
GPTA_SRC25 20-39
GPTA_SRC26 20-39
GPTA_SRC27 20-39
GPTA_SRC28 20-39
GPTA_SRC29 20-39
GPTA_SRC30 20-39
GPTA_SRC31 20-39
GPTA_SRC32 20-39
GPTA_SRC33 20-39
GPTA_SRC34 20-39
GPTA_SRC35 20-39
GPTA_SRC36 20-39
GPTA_SRC37 20-39
GPTA_SRC38 20-39
GPTA_SRC39 20-38
GPTA_SRC40 20-38
GPTA_SRC41 20-38
GPTA_SRC42 20-38
GPTA_SRC43 20-38
GPTA_SRC44 20-38
GPTA_SRC45 20-38
GPTA_SRC46 20-38
GPTA_SRC47 20-38
GPTA_SRC48 20-38
GPTA_SRC49 20-38
GPTA_SRC50 20-38
GPTA_SRC51 20-38
GPTA_SRC52 20-38
GPTA_SRC53 20-38
GPTA_SRS0 20-21
GPTA_SRS1 20-22
GPTA_SRS2 20-22
GPTA_SRS3 20-22
GPTU_CLC 20-15
GPTU_ID 20-15
GPTU_OSEL 20-15
GPTU_OUT 20-15
GPTU_SRC0 20-17
GPTU_SRC1 20-17
GPTU_SRC2 20-17
GPTU_SRC3 20-17
GPTU_SRC4 20-17
GPTU_SRC5 20-17
GPTU_SRC6 20-17
GPTU_SRC7 20-17
GPTU_SRSEL 20-16
GPTU_T012RUN 20-16
GPTU_T01IRS 20-15
GPTU_T01OTS 20-15
GPTU_T0CBA 20-16
GPTU_T0DCBA 20-16
GPTU_T0RCBA 20-16
GPTU_T0RDCBA 20-16
GPTU_T1CBA 20-16
GPTU_T1DCBA 20-16
GPTU_T1RCBA 20-16
GPTU_T1RDCBA 20-16
GPTU_T2 20-16
GPTU_T2AIS 20-15
GPTU_T2BIS 20-15
GPTU_T2CON 20-15
GPTU_T2ES 20-15
GPTU_T2RC0 20-16
GPTU_T2RC1 20-16
GPTU_T2RCCON 20-15

I

ICR 2-32, 14-7, 20-82

IOCONF 19-29
IOSR 19-31, 20-13
ISP 2-26, 2-32, 20-82

J

JPD_ID 20-13

L

LCX 2-25, 2-32, 20-82

M

MANID 4-9, 20-12
Memory protection system registers 11-4

N

NMISR 15-13, 20-11

O

OCDS module registers 19-8

P

P0_ALTSEL0 12-14, 20-52
P0_ALTSEL1 12-14, 20-52
P0_DIR 12-8, 20-51
P0_IN 12-7, 20-51
P0_OD 12-9, 20-51
P0_OUT 12-6, 20-51
P0_PICON 12-10, 20-52
P0_POCON 12-13, 20-52
P0_PUDEN 12-12, 20-52
P0_PUDSEL 12-11, 20-52
P1_DIR 12-8, 20-52
P1_IN 12-7, 20-52
P1_OD 12-9, 20-52
P1_OUT 12-6, 20-52
P1_PICON 12-10, 20-53
P1_POCON 12-13, 20-53
P1_PUDEN 12-12, 20-52
P1_PUDSEL 12-11, 20-52
P2_DIR 12-8, 20-53
P2_IN 12-7, 20-53
P2_OD 12-9, 20-53
P2_OUT 12-6, 20-53
P2_PICON 12-10, 20-53
P2_POCON 12-13, 20-53
P2_PUDEN 12-12, 20-53
P2_PUDSEL 12-11, 20-53
P3_DIR 12-8, 20-54
P3_IN 12-7, 20-54
P3_OD 12-9, 20-54
P3_OUT 12-6, 20-54
P3_PICON 12-10, 20-54
P3_POCON 12-13, 20-54
P3_PUDEN 12-12, 20-54
P3_PUDSEL 12-11, 20-54
P4_DIR 12-8, 20-54
P4_IN 12-7, 20-54
P4_OD 12-9, 20-54
P4_OUT 12-6, 20-54
P4_PICON 12-10, 20-55
P4_POCON 12-13, 20-55
P4_PUDEN 12-12, 20-55
P4_PUDSEL 12-11, 20-55
P5_ALTSEL0 12-14, 20-56
P5_DIR 12-8, 20-55
P5_IN 12-7, 20-55
P5_OD 12-9, 20-55
P5_OUT 12-6, 20-55
P5_PICON 12-10, 20-55
P5_POCON 12-13, 20-55
P5_PUDEN 12-12, 20-55
P5_PUDSEL 12-11, 20-55
PC 2-17, 2-32, 20-82
PCX 2-24
PCXI 2-22, 2-32, 20-82
PLL_CLC 3-9, 20-11
PMG_CON 6-4, 20-11
PMG_CSR 6-5, 20-11
PMU module registers 8-7
PMU_CON 8-8, 20-4
PMU_EIFCON 8-10, 13-32, 20-4
PMU_ID 20-4
Power management registers 6-3
PSW 2-18, 2-32, 11-7, 20-82
Px_...- Port kernel registers 12-4

R

Reset registers 5-2
RST_REQ 5-5, 20-11
RST_SR 5-3, 20-11
RTID 4-11, 20-12

S

SBSRC0 19-15, 20-78
SCU_CON 4-3, 20-12
SCU_ID 20-11
SCU_TRSTAT 4-8, 20-12
SSC0_BR 20-19
SSC0_CLC 20-19
SSC0_CON 20-19
SSC0_ESRC 20-20
SSC0_FSTAT 20-20
SSC0_ID 20-19
SSC0_RB 20-19
SSC0_RSRC 20-20
SSC0_RXFCON 20-20
SSC0_TB 20-19
SSC0_TSRC 20-20
SSC0_TXFCON 20-20
SSC0_WHBCON 20-20
SSC1_BR 20-20
SSC1_CLC 20-20
SSC1_CON 20-20
SSC1_ESRC 20-21
SSC1_FSTAT 20-21
SSC1_ID 20-20
SSC1_RB 20-21
SSC1_RSRC 20-21
SSC1_RXFCON 20-21
SSC1_TB 20-21
SSC1_TSRC 20-21
SSC1_TXFCON 20-21
SSC1_WHBCON 20-21
STM module registers 17-4
STM_CAP 17-6, 20-13
STM_CLC 17-7, 20-13
STM_ID 20-13
STM_TIM0 17-5, 20-13
STM_TIM1 17-5, 20-13
STM_TIM2 17-5, 20-13
STM_TIM3 17-5, 20-13
STM_TIM4 17-6, 20-13
STM_TIM5 17-6, 20-13
STM_TIM6 17-6, 20-13
SWEVT 19-13, 20-81
SYSCON 2-29, 2-32, 20-82

T

TR0EVT 19-14, 20-81
TR1EVT 19-14, 20-81

W

WDT module registers 18-28
WDT_CON0 18-29, 20-11
WDT_CON1 18-31, 20-11
WDT_SR 18-32, 20-11

Infineon goes for Business Excellence

“Business excellence means intelligent approaches and clearly defined processes, which are both constantly under review and ultimately lead to good operating results.
Better operating results and business excellence mean less idleness and wastefulness for all of us, more professional success, more accurate information, a better overview and, thereby, less frustration and more satisfaction.”

Dr. Ulrich Schumacher

http://www.infineon.com

Published by Infineon Technologies AG